14 March 2001 (14.03.2001)

10/018942

REC'D 02 MAY 2001

WIPO PCT

PA 371247

30/8

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

4

February 27, 2001

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE UNDER 35 USC 111.

CA01/214

APPLICATION NUMBER: *60/184,921*
FILING DATE: *February 25, 2000*

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH
RULE 17.1(a) OR (b)

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

H. L. Jackson

H. L. JACKSON
Certifying Officer

02-28-00

A/PROV

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s): OLIVER CRUDER and MICHAEL DAVID LOCKERBIE
Docket: 911.4USP1
Title: WIRELESS TELEPHONY INTERFACE

CERTIFICATE OF MAILING UNDER 37 CFR 1.10
Express Mail' mailing label number: EL307943118US
Date of Deposit: February 25, 2000
I hereby certify that this paper or fee is being deposited with the United States Postal Service 'Express Mail Post Office To Addressee' service under 37 CFR 1.10 and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.
By: Isabell Ogata
Name: Isabell Ogata

BOX PROVISIONAL PATENT APPLICATION
Assistant Commissioner for Patents
Washington, D.C. 20231

Dear Sir:

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 C.F.R. 1.53(c) on behalf of the following inventors:

(1)	**Full Name Of Inventor**	**Family Name** CRUDER	**First Given Name** OLIVER	**Second Given Name**
	Residence & Citizenship	**City** Saskatoon, Saskatchewan	**State or Foreign Country** CANADA	**Country of Citizenship** Canada
	Post Office Address	**Post Office Address** 502-611 University Drive	**City** Saskatoon, Saskatchewan	**State & Zip Code/Country** S7N 1Z3/CANADA
(2)	**Full Name Of Inventor**	**Family Name** LOCKERBIE	**First Given Name** MICHAEL	**Second Given Name** DAVID
	Residence & Citizenship	**City** Saskatoon, Saskatchewan	**State or Foreign Country** Canada	**Country of Citizenship** Canada
	Post Office Address	**Post Office Address** 419 Christopher Road	**City** Saskatoon, Saskatchewan	**State & Zip Code/Country** S7J 3S1/CANADA

Enclosed herewith are the following application parts:

☒ Transmittal sheet, in duplicate, containing Certificate Of Mailing Under 37 CFR 1.10.
☒ Provisional Patent Application: 149 page(s) of a specification.
☒ Our Check No. 30105206 in the amount of $150.00 to cover the Filing Fee.
☒ Return postcard.

Please charge any additional fees or credit overpayment to Deposit Account No. 50-0494 of Gates & Cooper. A duplicate of this sheet is enclosed.

CUSTOMER NUMBER 22462

GATES & COOPER
Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, CA 90045
(310) 641-8797

By: GHG/KPB
Name: George H. Gates
Reg. No.: 33,500
Initials: GHG/io

(PTO TRANSMITTAL - NEW FILING)

Docket: 911.4USP1

U.S. Provisional Patent Application for:

WIRELESS TELEPHONY INTERFACE

Inventors:

Oliver Cruder
502-611 University Drive
Saskatoon, Saskatchewan
S7N 1Z3 CANADA

Michael David Lockerbie
419 Christopher Road
Saskatoon, Saskatchewan
S7J 3S1 CANADA

CUSTOMER NUMBER 22462

GATES & COOPER
Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, California 90045
(310) 641-8797

EL307431185US

"Express Mail" mailing label number EL307431185US
Date of Deposit [illegible]
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

ISABELL OGATA
(Printed Name)

Isabell Ogata
(Signature)

The TelePATH is a unique product that aims to provide an alternative to the physical wire lines traditionally used to connect public payphones to the central office (CO) of the public switched telephone network (PSTN). The primary benefit of the TelePATH over conventional wired systems is that it can provide service to locations once considered too remote, inaccessible or environmentally hostile. The TelePATH can also be used in temporary installations. The TelePATH provides a full-featured, transparent wireless connection between a telephone and the wire back to the central office.

The TelePATH also has marketable advantages when compared to wireless alternatives, particularly cellular payphone applications. Wireless cellular applications are restricted to areas where coverage (i.e. cellular infrastructure) is available. The TelePATH is a self-contained system that essentially replaces the wired connection between the public payphone and the central office with a point-to-point wireless link and therefore provides coverage wherever it is needed. Further, the TelePATH supports loop polarity answer supervision. This is the traditional method employed in wired telephone installations; in contrast, cell- and cordless-payphone solutions must rely on less reliable approaches.

The preferred embodiment of the TelePATH operates in the 900 MHz ISM (Instrumentation, Scientific, and Medical) frequency band. Therefore, within certain power levels, a radio license is not required for operation in most jurisdictions, resulting in significant cost savings and added convenience over devices operating in other frequency bands. However, it need not be so limited.

The TelePATH has been designed to be virtually transparent to the phone system. Though wireless, it appears to both the central office and phone as a pair of wires connecting the two sides. Because of its transparency, the TelePATH does not need to interpret dialing digits; instead, it simply passes any in-band signal (voice, modem, music, DTMF tones, etc.) as audio between the central office (CO) and telephone just like a wire. Further, out-of-band signals (including hook status, loop polarity, and ringing) are also passed between the telephone and central office just like a wire. As will be explained in greater detail hereinafter, in-band signals are encoded in

ADPCM (adaptive differential pulse code modulation), while on-of-band signals are binary coded and transmitted with every frame.

Wireless range is related to transmitted power levels which are governed by regulatory agencies (FCC in the US and IC in Canada). Unlicensed narrow-band transmission power is restricted severely which can be detrimental to range. Spread spectrum techniques are allowed much higher transmission levels and could aid in extending range.

Payphone call privacy is important so the invention should employ voice encryption or "stealthy" transmission. Spread-spectrum techniques can provide the stealthy operation.

Commercial communications equipment (payphones) should provide reliable, robust service; therefore, the invention should be insensitive to interference and multi-path effects. Spread-spectrum techniques offer improved performance over narrow-band methods. In the preferred embodiment, Direct Sequence Spread Spectrum (DSSS) is used, which chops the data into small pieces and spreads them across the frequency domain. DSSS offers high immunity to interference and multipath effects.

Commercial communications equipment (payphones) should provide high-quality voice clarity; therefore, the invention should employ digital coding of voice signals. Many such codings are available, but in the preferred embodiment, the invention employs ADPCM (adaptive differential pulse code modulation) and in particular, 32 kbit per second ADPCM, which has been standardized under CCITT standard G721.

Adaptive Differential Pulse Code Modulation (ADPCM) is a waveform coding in which the difference between the speech signal and a prediction that has been made of the speech signal is coded, rather than quantizing the speech signal directly, like PCM. If the prediction is accurate then the difference between the real and predicted speech samples will have a lower variance than the real speech samples, and will be accurately quantized with fewer bits than would be needed to quantize the original speech samples. The performance is aided by using adaptive prediction and quantization, so that the predictor and difference quantizer adapt to the changing characteristics of the speech being coded. This coding gives reconstructed speech almost as good as 64 kbit per second PCM (Pulse Code Modulation) coding.

The TelePATH Millennium has been designed to be as transparent to the phone system as is possible. It monitors the hook-, loop polarity-, and ringing-status at the respective end and relays this information digitally to the mating device which emulates that state. While off hook, it relays digital voice signals full duplex.

Because of its transparency, the system does not need to interpret dialling digits. The hook status delay through the system (from the phone to the central office) is approximately 40 ms so that dial-tone is presented immediately upon going off-hook. Currently, we do not deny loop current if the wireless link is down, though it could easily be. If the wireless link is down, no dial tone will be present to the user when going off-hook.

The system supports the following:

- Two wire loop start
- DTMF or pulse-dialling
- Loop reversal
- 1200 baud (Bell 212)
- V.32bis, V.42 data compression and error correction
- Eight co-located systems without interference (ganged installations)

The line-interface side looks to the central office (CO) of the public switched telephone network (PSTN) like a loop-start telephone with the following parameters:

- > 600 Ohm AC impedance
- < 1 REN
- 40 to 120 VAC ring detection

The phone-interface side looks to the telephone like a loop-start central office (CO) with the following parameters:

- 600 Ohm AC impedance
- 25 mA loop current max (could be increased)
- 12 mA on/off hook detection threshold
- > 40 VACrms 20 Hz sinusoidal ringing voltage at 1 REN (approx. 47 VACrms into a Nortel Millennium)
- -48 Vdc battery voltage

Hence, any device which can be plugged into a standard telephone outline can also be connected to the TelePATH, including for example: pulse and Touch-Tone™ or DTMF (dual tone multi-frequency) telephones, cordless telephones, computer modems or facsimile machines. As described above, the preferred embodiment is the application to payphone telephones.

The TelePATH digital wireless link offers a cost effective solution for point to point voice and data communication.

The TelePATH / Millenium has been designed specifically for the Nortel Millenium payphone and provides a full featured, transparent wireless connection. This system provides a robust, reliable and secure toll-quality wireless connection between a public telephone and the wire connection back to the central office. The Millenium telephone, for example, is operable to communicate status data back to the central office. The TelePATH encodes these data into 14.4 kbps digital modem format to communicate over the wireless link.

Offering maximum flexibility in the deployment of public telephone, the TelePATH / Millenium is effective for both temporary installation and sites where it is impractical to run wires due to man made or natural obstacles.

Benefits:

- reduce installation cost;
- reduce installation time;
- portable public telephone access;
- effortlessly adapt to changes in site layout;
- ISM band for license free operation;
- digital spread spectrum for quality, security and reliability; and
- distances from 100m to 5000m.

Applications:

- construction sites;
- agricultural use, such as house to barn;
- sporting events;

- remote locations;
- exhibitions;
- demonstrations;
- institutional public phone;
- site testing and evaluation; and
- wireless last mile.

While particular embodiments of the present invention have been shown and described, it is clear that changes and modifications may be made to such embodiments without departing from the true scope and spirit of the invention.

The method steps of the invention may be embodiment in sets of executable machine code stored in a variety of formats such as object code or source code. Such code is described generically herein as programming code, or a computer program for simplification. Clearly, the executable machine code may be integrated with the code of other programs, implemented as subroutines, by external program calls or by other techniques as known in the art.

The embodiments of the invention may be executed by a computer processor, ASIC or similar device programmed in the manner of method steps, or may be executed by an electronic system which is provided with means for executing these steps. Similarly, an electronic memory medium such a computer diskette, CD-Rom, Random Access Memory (RAM), Read Only Memory (ROM) or similar computer software storage media known in the art, can store code which may be executed to perform such method steps. As well, electronic signals representing these method steps may also be transmitted via a communication network.

It would also be clear to one skilled in the art that this invention need not be limited to the communication devices described herein.

WHAT IS CLAIMED IS:

1. A communications interface comprising:

means for receiving electrical signals from a standard telephony device; and

means for converting and transmitting said received signals in a wireless frequency band.

2. A communications interface comprising:

means for receiving wireless transmitted signals; and

means for converting said transmitted signals into electrical signals for a standard central office.

3. A method of communication between a standard telephony device and a central office comprising the steps of:

receiving electrical signals from a standard telephony device; and

converting and transmitting said received signals in a wireless frequency band.

4. A method of communication between a standard telephony device and a central office comprising the steps of:

receiving wireless transmitted signals; and

converting said transmitted signals into electrical signals for a standard central office.

5. A computer readable memory medium, storing computer software code executable to perform the steps of the method claimed in either of claims 3 or 4.

6. A computer data signal embodied in a carrier wave, said computer data signal comprising a set of machine executable code being executable by a computer to perform the steps of the method claimed in either of claims 3 or 4.

CRITICAL CONTROL CORP.
TelePATH Millennium System Description

Document #: 18-03 Customer: Critical Control Corp. Original Project #: 990712

Release	Date	DCN #	Change	Approved	Customer
1.0	99.12.22		Initial release	MDL	

1.

OVERVIEW

The TelePATH Millennium provides a wireless link between a public telephone and its wire connection back to the central office. The system consists of two interface enclosures, two antennas, and two coaxial antenna cables. The "Line Interface" connects to the central office line. The "Phone Interface" connects to the phone. Figure xxx illustrates a pictorial of the system.

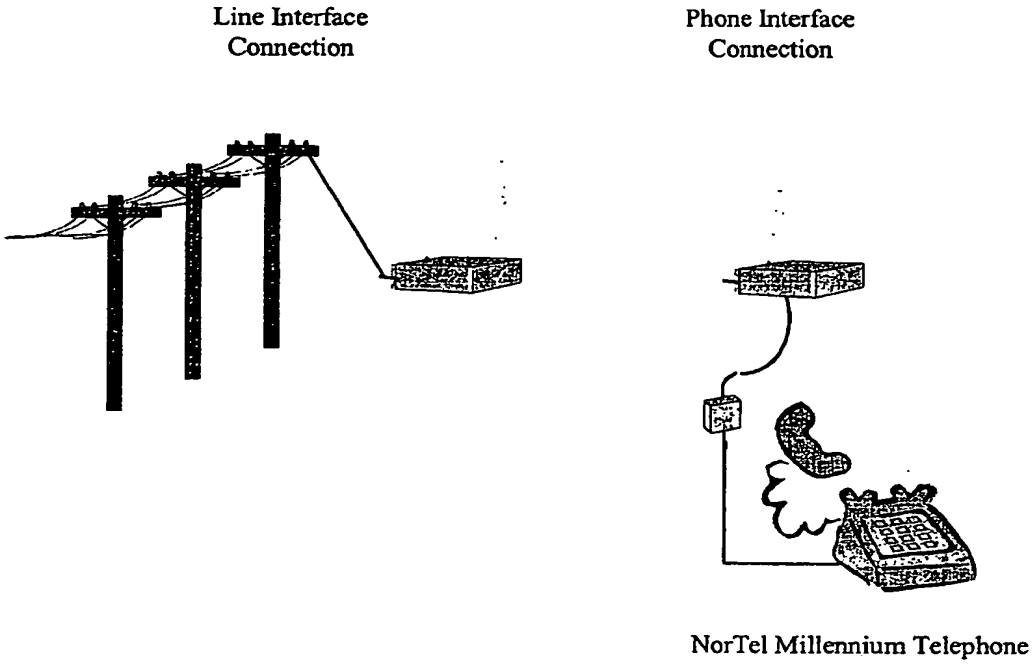


Figure 1: System Topology

2.

2. HARDWARE

The hardware of the line- and phone-interface devices are sufficiently similar as to allow a single diagram to illustrated both devices. Both devices are identical by design with the exception of the the *T/R Interface* section which deals with the specific interfacing requirements of the phone and line sides. Figure 1 shows a block diagram of all sections of the TelePATH Millennium. Each block is described below.

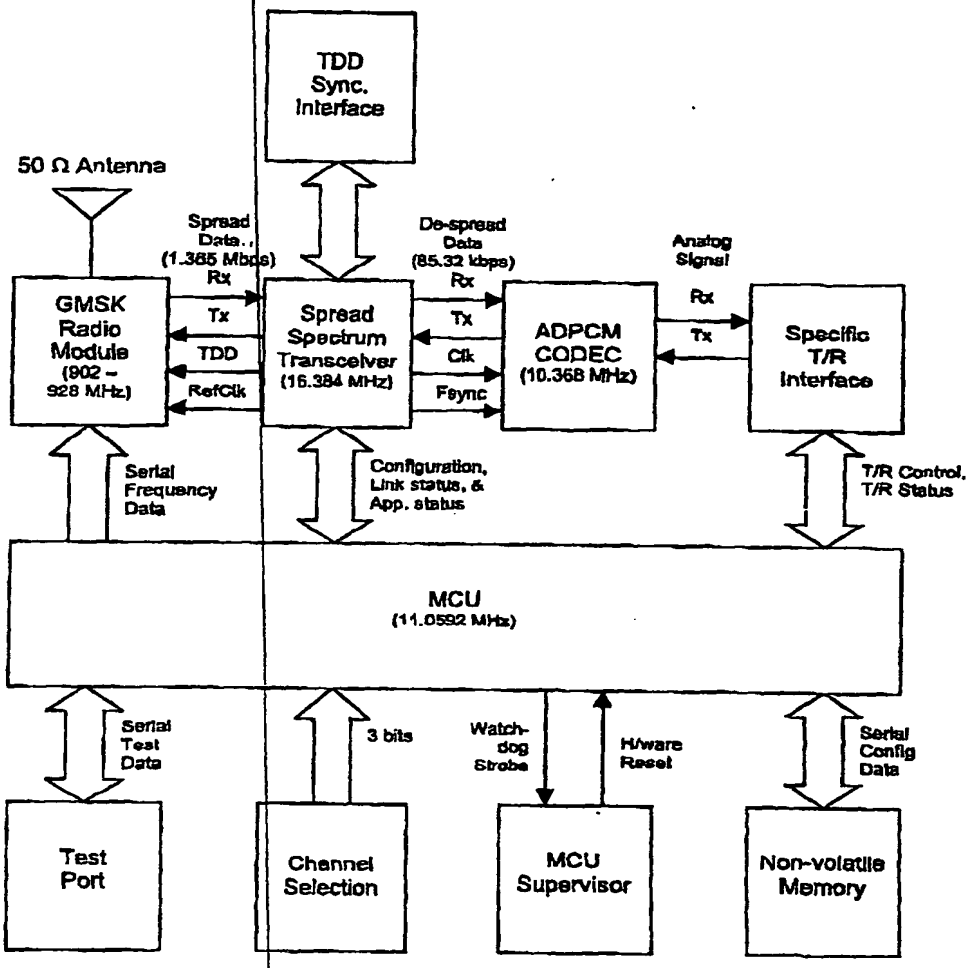


Figure 2: Hardware Block Diagram

MCU

The *MCU* section is composed of an 8-bit microcontroller that drives an 11.0592 MHz crystal. The microcontroller controls all aspects of device operation including oscillator frequency and PN sequence selection but it is not directly involved with data modulation.

2.2 Test Port

The *Test Port* section provides a serial link (19.2 kbps maximum) between the TelePATH and a computer. This port is used during device configuration and testing phases.

2.3 Channel Select

The *Channel Select* section allows user input for selection of one of eight communications channels.

2.4 MCU Supervisor

The *MCU Supervisor* section monitors power supply quality and MCU operation and provides a hardware reset signal when appropriate.

2.5 Non-volatile Memory

The *Non-volatile Memory* section provides a non-volatile memory space for configuration parameter storage and retrieval.

2.6 Gaussian Mean Shift-Key (GMSK) Radio Module

The *GMSK Radio Module* is a complete radio transceiver module. The transmit chain filters base-band data with a Gaussian spectral shape. The resulting signal then directly modulates the VCO in the 902 – 928 ISM band. The receiver chain performs down-conversion of RF to IF (44 MHz) and then demodulates the GMSK IF to base-band. The module operates in time-division duplex mode (TDD) with a 9 ms cycle time and so cannot transmit and receive simultaneously. The on-board VCO accepts its 16.384 MHz reference clock from the Spread Spectrum Transceiver module. The GMSK radio module functional block diagram is shown in Figure 2.

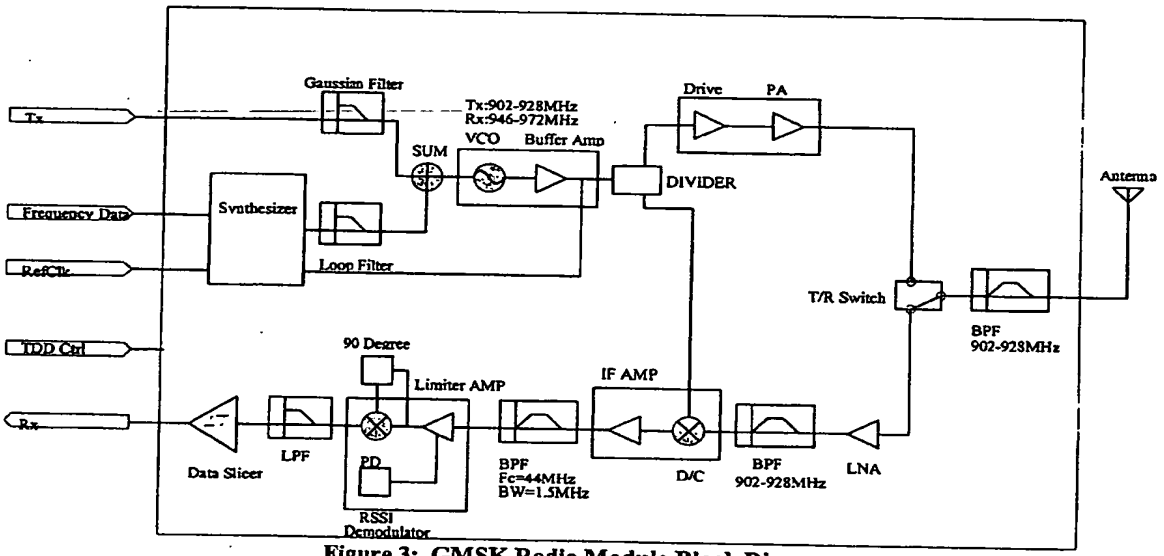


Figure 3: GMSK Radio Module Block Diagram

2.7

Spread Spectrum Transceiver

The *Spread Spectrum Transceiver* module performs a variety of functions including TDD control, data spreading/de-spreading, reference clock generation, and radio and ADPCM codec interfacing.

The TDD controller implements a TDD protocol that allows a full-duplex link to be emulated by the half-duplex radio. It also generates the TDD control signals and the frame-synchronization signals required by the GMSK Radio Module and ADPCM Codec respectively. The controller buffers the codec data in order to convert between the 32 kbps full duplex codec data stream and the 85.33 kbps half-duplex on-air data rate used in the TDD scheme. The controller uses a digital phase locked loop to maintain an equal read and write rate to the rate buffers to avoid FIFO over/under-flow.

Transmitter and receiver logic spreads/de-spreads the data to accommodate the on-air chip rate of 1.365 Mbps. The PN sequence for each symbol is programmed by the microprocssor according to the selected channel. The long PN sequence is fixed in hardware. The receiver samples the incoming baseband signal at two samples per PN chip. The samples are correlated with the four possible PN sequences in 64-bit parallel correlators. The de-correlated signal is demodulated via a digital phase locked loop.

The transceiver generates both the 16.384 MHz reference clock required by the radio and the 2.048 MHz serial clock required by the codec.

The transceiver multiplexes and de-multiplexes overhead bits with the voice data which are required for link maintenance.

2.8 ADPCM Codec

The *ADPCM Codec* section comprises a single channel ADPCM codec. It performs mutual transcoding between the 300 to 3400 Hz analog tip and ring signal and a 32 kbps ADPCM full-duplex serial data stream. The codec accepts synchronous serial clock and frame-synchronization signals from the spread spectrum transceiver.

2.9 T/R interface

The *T/R interface* section interfaces the specific tip and ring circuitry to both the ADPCM codec and the microcontroller. It is unique for both the phone- and line-interface devices.

3. MODULATION

3.1 Data Frames

One device of the linked pair is designated the TDD master, and the other the slave. Master and slave each have unique roles in the TDD protocol. The master initiates the communication link with the slave. Four frame formats are used during the set-up and maintenance of the TDD communications link. Initially, the two communicating devices need to establish "sync". The TDD protocol achieves this by using a special handshaking protocol. The master device first transmits an "acquisition burst". The acquisition burst consists of 32 bits of preamble (binary 0's), followed by 226 bits of "zero stuffing", and four 22-bit unique words (UW). When the slave device receives the acquisition burst from the master correctly (by decoding the 4 consecutive UW's) it sends an acquisition burst in response. When the master receives the acquisition burst, it sends an "empty burst". An empty burst contains a 32-bit preamble followed by a single 22-bit unique word, and 292-bit of "1" (One-stuffing). In response to the master's empty burst, the slave also sends an empty burst back to the master. When the master receives the empty burst from the slave, the communication link is considered to have been established and the "sync" condition achieved. On the following burst, both the master and the slave start genuine data transmission by sending out "data bursts". Each of the data bursts contains a 32-bit preamble, followed by a 22-bit UW, a 4-bit status nibble (ST), and 288 bits of user data (ADPCM voice samples or data). The three different types of burst frame structures are shown in Figure 3.

Each actual burst cycle also includes two guard times to allow for both propagation and RF transceiver switching time. More specifically, G1 is a 32-bit delay between the time when the master stops transmission and the slave commences transmission; G2 is a 32-bit delay between the time when the slave stops transmission and the master commences transmission as shown in Figure 4. These guard times allow for a 375 µs delay. The total burst cycle is 768 bits long, including 12 bits internal delay (transmitter turns off 6 bits after the last data bit is latched into the transmitter, the master and slave therefore contribute a total of 12-bit internal delay).

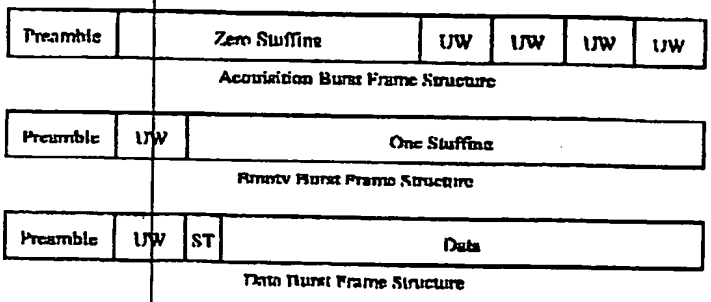


Figure 4: Burst Frame Structure

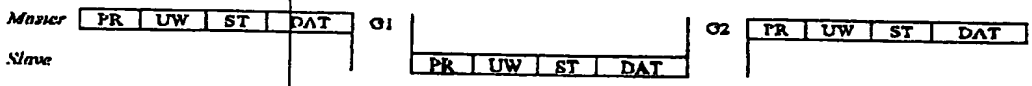


Figure 5: Master/Slave Frame Relationship

3.2 Direct Sequence Spread Spectrum

Transceiver logic spreads the 85.33 kbps data frames with a chip rate of 1.365 Mbps to achieve a 12 dB process gain. Consecutive data frame bit-pairs are encoded as one of four symbols each with a unique 32-bit PN sequence. Data is further randomized by modulus-2 addition with a 2047-bit PN sequence. This operation smoothes the output spectrum of the transmitted signal and eliminates discrete spectral components.

Each of eight channels corresponds to a unique set of four PN sequences. All PN sequences are listed in Table 1.

Table 1: PN Sequences

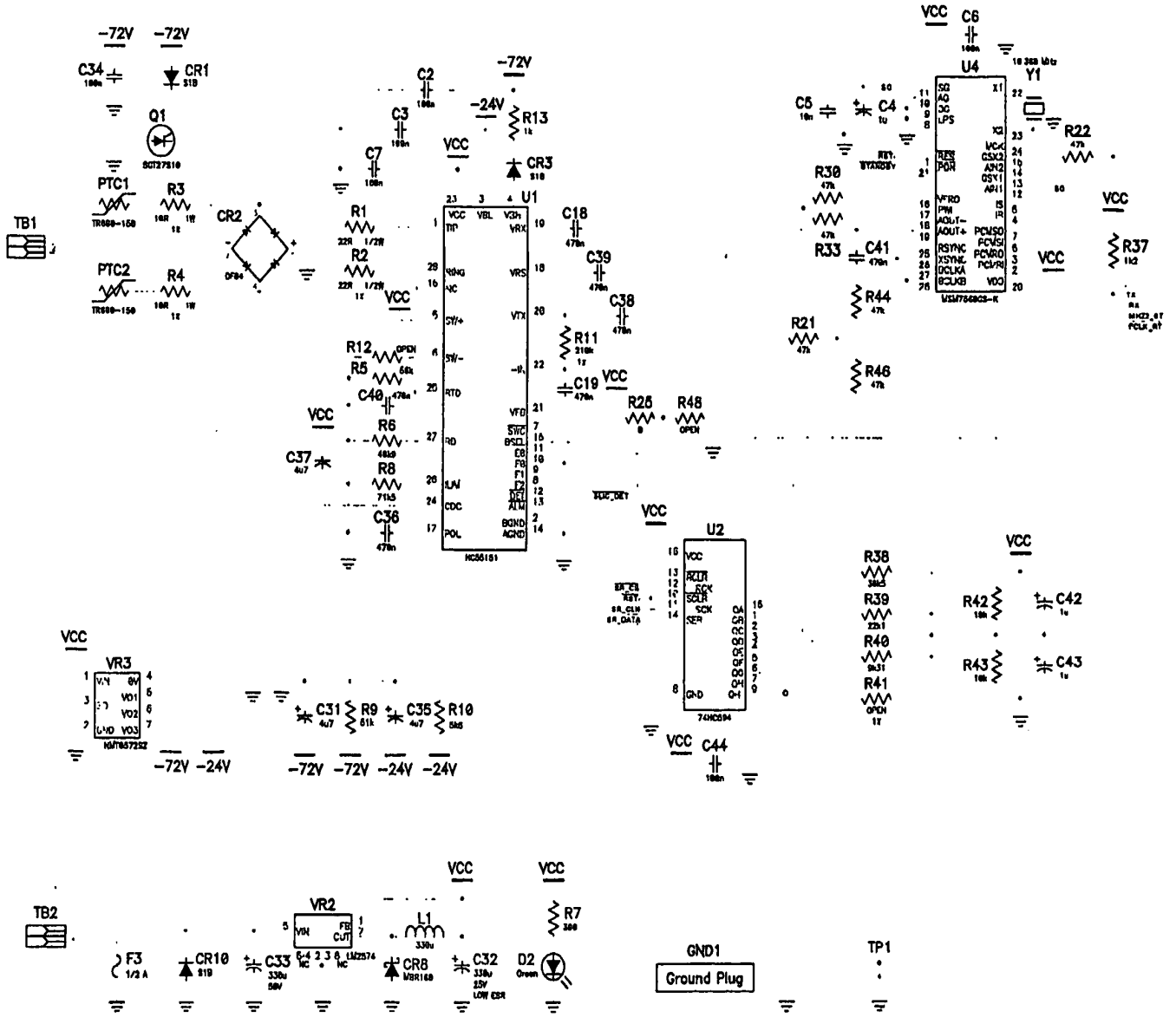
Channel	Symbol			
	A	B	C	D
0	0xD6AD88D6	0x5598D6A5	0x96CAF149	0x67396869
1	0x68CAA59E	0xDC8F4654	0xF1A8CBA4	0xF4405D7A
2	0x8C3CF515	0xA153ACD5	0X77066437	0xC18A55ED
3	0xB9ADEBD8	0xC61B7A8A	0x4DF29B0C	0x1368D79A
4	0x78D465D2	0xAC5AD2B2	0xC4823E50	0x655D9D14
5	0x50BAA739	0xBB83321B	0x42A759AB	0x8CE2E3C3
6	0x054C5513	0xBEA24F87	0xD435C92B	0x4F5168B5
7	0x83E80A70	0xF33C8196	0x129596FA	0x087A249A

3.3Radio Transceiver

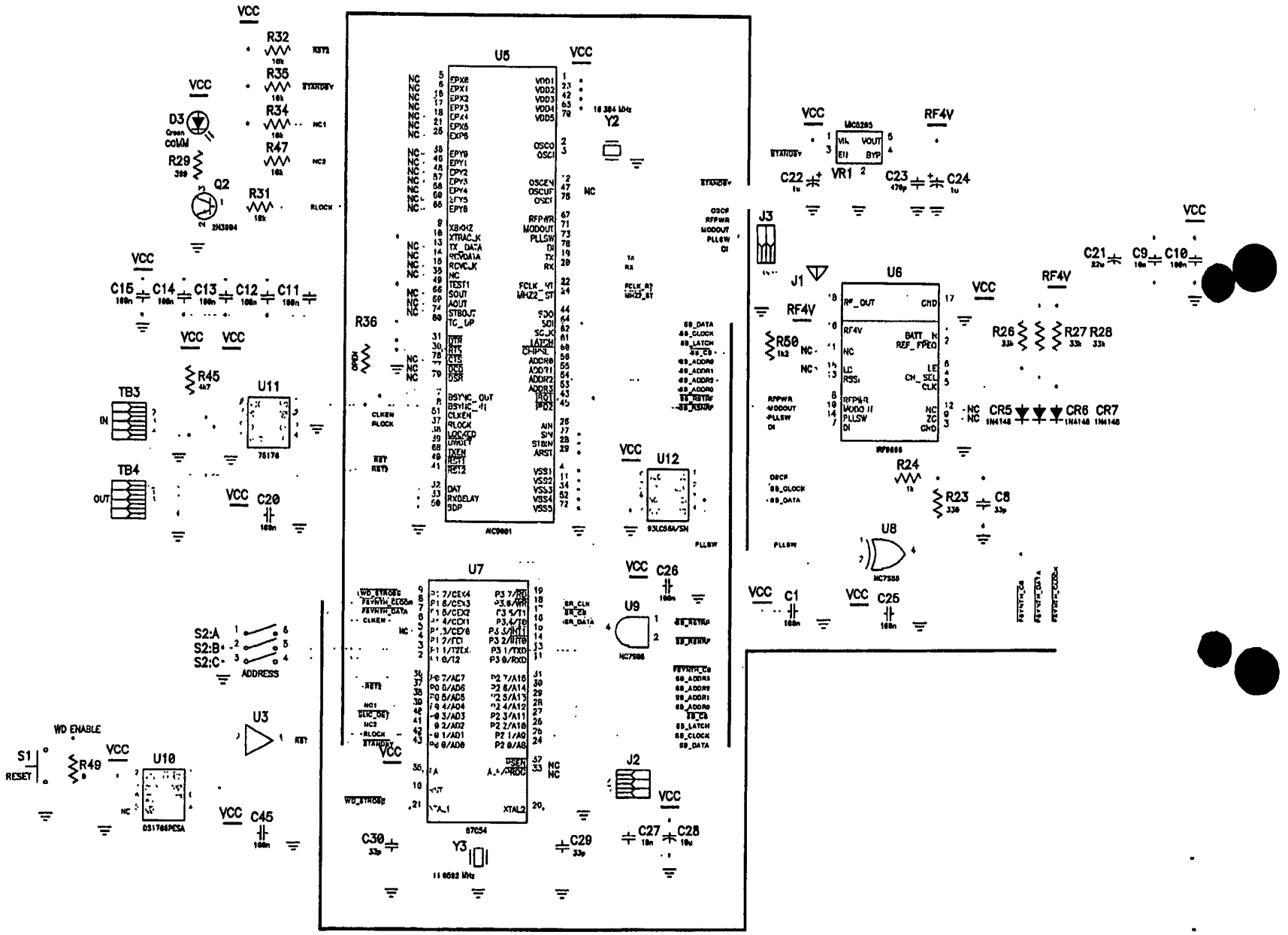
The spread signal, filtered with a Gaussian spectral shape, directly modulates the RF carrier. Each of eight channels has a unique carrier frequency. The radio receiver comprises a super-heterodyne topology with an intermediate frequency (IF) of 44 MHz. The local oscillator (LO) tunes the lower side-band (LSB) to the IF for subsequent GMSK demodulation. Carrier and LO frequencies are listed in table 2.

Table 2: VCO Frequencies

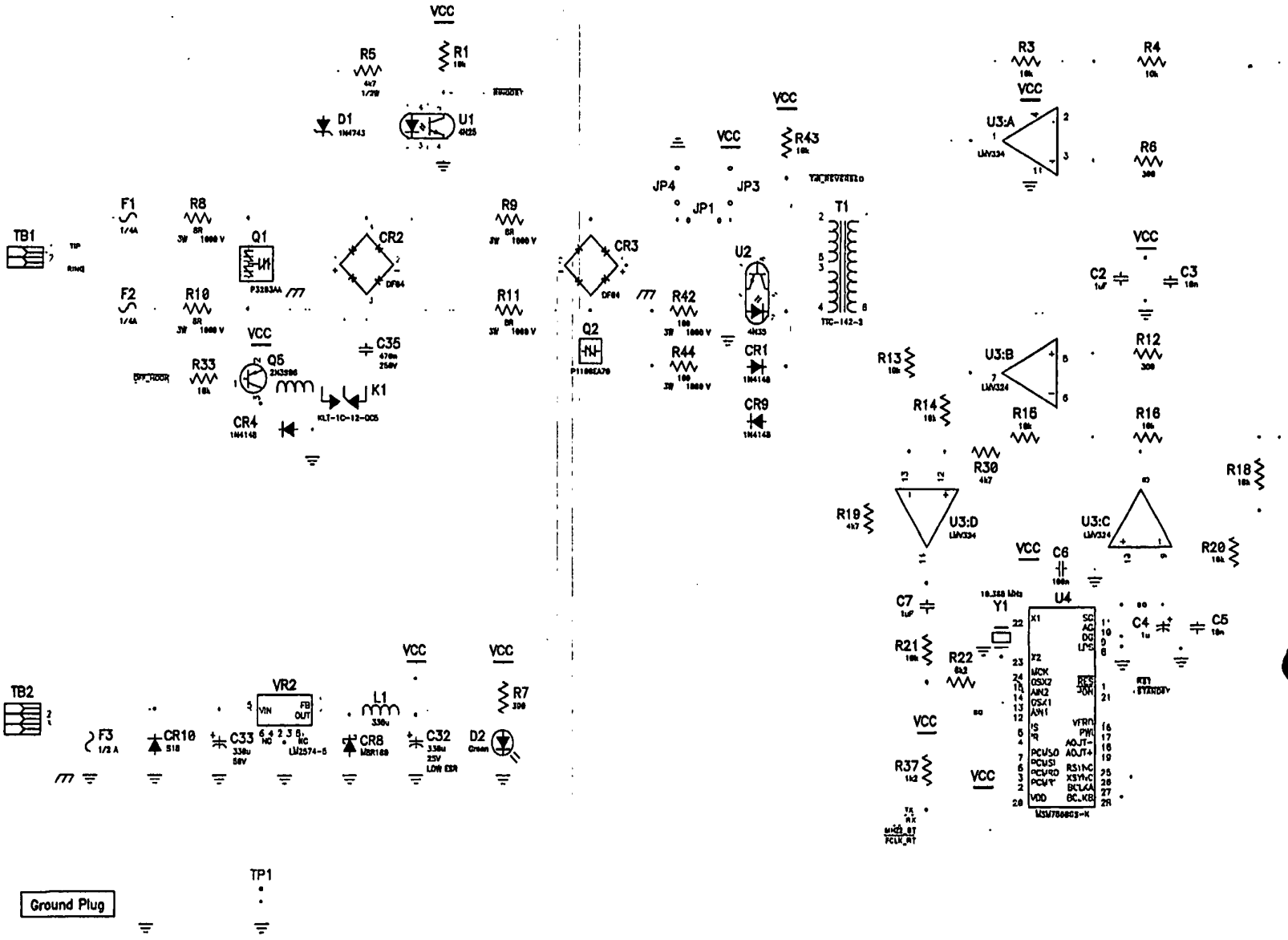
Channel	VCO Frequency	
	Transmit Carrier [MHz]	Receive LO [MHz]
0	924.928	968.960
1	922.624	966.656
2	920.064	964.096
3	917.504	961.536
4	914.944	958.976
5	912.640	956.672
6	910.336	954.368
7	908.032	952.064



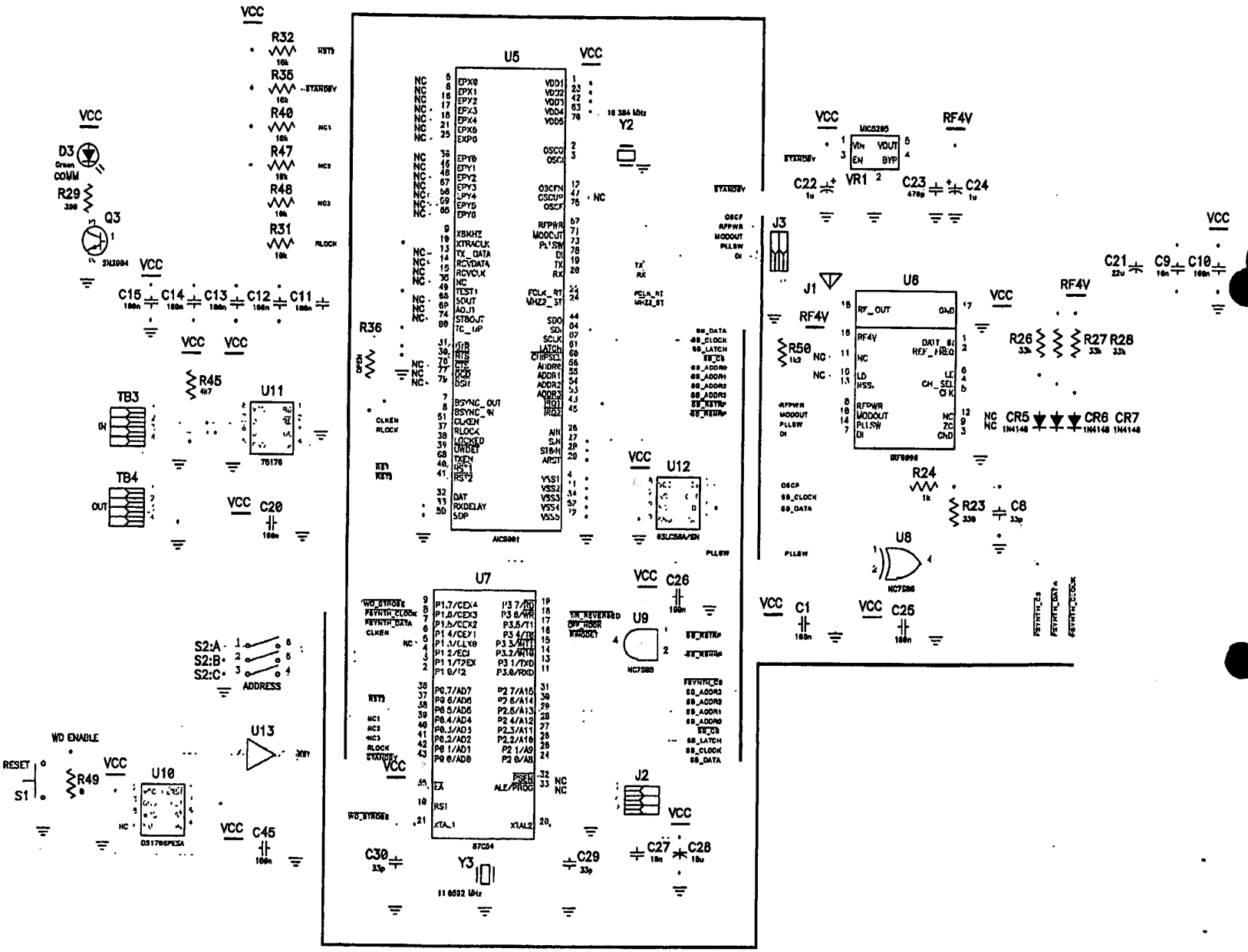
Phone Schematic Page 1



Phone Schematic Page 2



Line Schematic Page 1



Line Schematic Page 2

TelePATH Millennium Features

Michael D. Lockerbie
00.01.06

The TelePATH Millennium has been designed to be as transparent to the phone system as is possible. It monitors the hook-, loop polarity-, and ringing-status at the respective end and relays this information digitally to the mating device which emulates that state. While off hook, it relays digital voice signals full duplex.

Because of its transparency, the system does not need to interpret dialling digits. The hook status delay through the system (from the phone to the central office) is approximately 40 ms so that dial-tone is presented immediately upon going off-hook. Currently, we do not deny loop current if the wireless link is down (could be changed). If the wireless link is down, no dial tone will be present to the user when going off-hook.

The system supports the following:

- Two wire loop start
- DTMF or pulse-dialling
- Loop reversal
- 1200 baud (Bell 212)
- V.32bis, V.42 data compression and error correction
- Eight co-located systems without interference (ganged installations)

The line-interface side looks to the central office like a loop-start phone with the following parameters:

- > 600 Ohm AC impedance
- < 1 REN
- 40 to 120 VAC ring detection

The phone-interface side looks to the phone like a loop-start central office with the following parameters:

- 600 Ohm AC impedance
- 25 mA loop current max (could be increased)
- 12 mA on/off hook detection threshold
- > 40 VACrms 20 Hz sinusoidal ringing voltage at 1 REN (approx. 47 VACrms into a Nortel Millennium)
- -48 Vdc battery voltage

ITEM	QTY REQ	PART NO	DESCRIPTION	DETAIL	REFERENCE DESIGNATORS
1	1	02-0.5-2AG	Fuse	1/2 Amp, 2AG	F3
2	2	02-100056	Fuse Clip	2AG	for F3
3	1	04-18-10-REV4	PCB	WPOTS Phone Board	PCB
4	2	05-502G	Terminal Block	2 position, str., 0.200" sp., Grey	TB1, TB2
5	1	05-504G	Terminal Block	4 position str. 0.200" sp., Grey	TB3
6	1	05-MMS-102	2 Pin Single Socket	2mm, 2 x 1	JSH
7	1	05-MMS-108	16 Pin Dual Socket	2mm, 8 x 2	JDH
8	0.125	06-40-01S	Strip Header	40 Pin SIL, 0.1"	RSSI (2), J2 (3)
9	1	06-PLCC44	PLCC Socket	44 Pin	for U7
10	1	13-0301	Switch	3-position DIP	S2
11	1	13-1000	Switch	Omron NO, momentary	S1
12	1	14-337	Inductor	330uH Linear	L1
13	1	20S-87C52	SMT 8-bit Microcontroller	16 MHz, PLCC44, Industrial, OTP	U7
14	1	21S-10368-IND	SMT Crystal	10.3680 MHz, Industrial, HC49SD	Y1
15	1	21S-110592-IND	SMT Crystal	11.0592 MHz, Industrial, HC49SD	Y3
16	1	21S-16384-IND	SMT Crystal	16.3840 MHz, Industrial, HC49SD	Y2
17	1	22S-93C66	EEPROM SMT	Microchip, Industrial Temp, 8 pin SOIC	U12
18	1	23S-74HC594	74HC594	8-Bit Shift Register, Industrial, SOIC 150 mil	U2
19	1	23S-75176	Transceiver	RS-485	U11
20	1	23S-DS1706P	MicroMonitor	3.3 Volt, Industrial Temp, 8-pin SOIC	U10
21	1	23S-HC55181	Ringing Subscriber Line Interface Circuit	Polarity Reversal, 28 pin PLCC, Industrial	U1
22	1	23S-MSM7560	ADPCM CODEC	Single Rail, u-Law, SOP28	U4
23	1	23S-NC7S04	Single Inverter	Tiny Logic, 5 pin SOT-23	U3
24	1	23S-NC7S08	Single AND gate	Tiny Logic, 5 pin SOT-23, -40 to +85	U9
25	1	23S-NC7S86	Single XOR gate	Tiny Logic, 5 pin SOT-23	U8
26	1	24-IRF9096	PCBA RF-9096	RF Module IRF9096DS 900MHz 96Kbps	U6
27	1	24S-AIC9001	IC, SST ASIC	AIC9001, PQFP-80	U5
28	1	31-NMT0572SZ	DC - DC Converter	5V IN, -24V, -48V, -72V OUT	VR3
29	1	31S-3904-SOT23	SMT NPN 3904 Transistor		Q2
30	1	32-2574-5	Voltage Regulator	Switching Reg. 5V	VR2
31	1	32-DF04M	Bridge Rectifier 1A, 400V		CR2
32	1	32S-MIC5205	4V Voltage Regulator	Industrial Temp, SOT23-5	VR1

ITEM	QTY	PART NO	DESCRIPTION	DETAIL	REFERENCE
33	1	34-160	Shottky Diode	1.0A 60V	CR8
34	3	34S-4148-SOT23	Switching Diode	1N4148 SOT23	CR5,CR6,CR7
35	3	34S-S1B	SMT General Purpose Diode	100V 1.0A SMA Package	CR1, CR3, CR10
36	2	36-4234	LED	Green, T 1-3/4	D2, D3
37	2	38-PTC-600	Polyswitch	60V, 600V interrupt, 3A	PTC1, PTC2
38	1	38-SGT27S10	Transient Surge Suppressor	unidirectional, gate controlled	Q1
39	1	41-104-250	Capacitor	0.1uF 250 Vdc metalized poly., 0.4"	C34
40	1	41-337-25ESR	Capacitor	330uF 25V Radial Low ESR	C32
41	1	41-337-50	Capacitor	330uF 50V Radial 0.2"	C33
42	1	41-475-100	Capacitor	4.7uF, 100V, Aluminum, -40° to +85°C	C31
43	1	41-475-63	Capacitor	4.7uF, 63V, Aluminum, -55° to +105°C	C35
44	3	41S-103-603	SMT Capacitor 0603, X7R	10nF	C5, C9, C27
45	2	41S-104-1206-100	SMT Chip Capacitor 1206	100nF 10% 100V	C2, C3
46	14	41S-104-603	SMT Chip Capacitor, X7R	100nF, 25V +/-10%	C1, C6, C7, C10, C11, C12, C13, C14, C15, C20, C25, C26, C44, C45
47	5	41S-105-16	SMT Capacitor, tant.	1uF 16V, -55 to +125 C, 3216 Size	C4, C22, C24, C42, C43
48	1	41S-106-3528-16	SMT Capacitor EIA std 3528	10uF 16V tant. -55 to +125 C	C28
49	1	41S-226-16-B	SMT Polarized Capacitor B type (3528)	22uF, 16V, -55 to +125 C, 10%	C21
50	3	41S-330-603	SMT Capacitor 0603, NPO	33pF, 50V	C8, C29, C30
51	1	41S-471-603	SMT Capacitor 0603, X7R	470pF, 50V	C23
52	7	41S-474-1206	SMT Capacitor, tant.	470nF 25V, -55 to +125 C, 10%	C18, C19, C36, C38, C39, C40, C41
53	1	41S-475-3528	SMT Capacitor EIA std 3528	4.7uF 16V tant. -55 to +125 C	C37
54	2	51-100-1005	Resistor	10R 5% 1W Metal Film	R3, R4
55	1	51-102-0255	Resistor	1K0 5% 1/4W	R13
56	2	51-220-0501	Resistor	22R 1% 0.6W Metal Film	R1, R2
57	1	51-513-0255	Resistor	51K 5% 1/4W	R9
58	1	51-562-0255	Resistor	5K6 5% 1/4W	R10
59	2	51S-000-603	SMT Resistor 0603	0 Ohm 5%, 1/16 W	R49, C+
60	1	51S-102-603	SMT Resistor 0603	1 KOhm 5%, 1/16 W, 50V	R24
61	5	51S-103-603	SMT Resistor 0603	10K Ohm 5%, 1/16 W, 50V	R31, R32, R34, R35, R47
62	1	51S-122-603	SMT Resistor 0603	1K2 Ohm 5%, 1/16 W, 50V	R37, R50
63	1	51S-2103-603	SMT Resistor 0603	210k 1% 1/16 W	R11
64	1	51S-2212-603	Chip Resistor 0603	22k1 1%	R39
65	1	51S-331-603	SMT Resistor 0603	330 Ohm 5%, 1/16 W, 50V	R23
66	1	51S-333-603	SMT Resistor 0603	33 kOhm 5%, 1/16 W, 50V	R26, R27, R28
67	1	51S-3652-603	Chip Resistor 0603	36k5 1%	R38
68	2	51S-391-603	SMT Resistor 0603	390 Ohm 5%, 1/16 W, 50V	R7, R29
69	1	51S-472-603	SMT Resistor 0603	4.7 KOhm 5%, 1/16W	R45
70	6	51S-473-603	SMT Resistor 0603	47 KOhm 5%, 1/16 W, 50V	R21, R22, R30, R33, R44, R46
71	1	51S-4992-603	SMT Resistor 0603	49k9 1% 1/16W	R6
72	1	51S-563-603	SMT Resistor 0603	56k 5% 1/16W	R5

ITEM	QTY	PART NO	DESCRIPTION	DETAIL	REFERENCE
73	1	51S-7152-603	SMT Resistor 0603	71k5 1% 1/16 W	R8
74	2	51S-822-603	SMT Resistor 0603	8k2 5% 1/16 W	R42, R43
75	1	51S-9311-603	SMT Resistor 0603	9K31 Ohm 1%, 1/16 W	R40
76	1	90-18-08-REV3	ASSEMBLY	PWA Assembly	Assembly

ITEM	QTY REQ	PART NO	DESCRIPTION	DETAIL	REFERENCE DESIGNATORS
1	1	02-0.5-2AG	Fuse	1/2 Amp, 2AG	F3
2	2	02-1/4A-TELCO	Fuse	1/4A 2AG Surge withstand 250V	F1,F2
3	6	02-100056	Fuse Glip	2AG	for F1, F2, F3
4	1	04-18-05-REV4	PCB	WPOTS Line Board, Rev 4	PCB
5	1	05-502G	Terminal Block	2 position, str., 0.200" sp., Grey	TB1
6	1	05-503G	Terminal Block	3-position, str., 0.200" sp., Grey	TB2
7	1	05-504G	Terminal Block	4 position str. 0.200" sp., Grey	TB3, TB4
8	1	05-MMS-102	2 Pin Single Socket	2mm, 2 x 1	JSH
9	1	05-MMS-108	16 Pin Dual Socket	2mm, 8 x 2	JDH
10	0.225	06-40-01S	Strip Header	40 Pin SIL, 0.1"	RSSI (2), J2 (3), Test Header (4)
11	1	06-4051-60	Jumper		Test Header (2)
12	1	06-PLCC44	PLCC Socket	44 Pin	for U7
13	1	11-142-2	Transformer Tamura	Phone Line	T1
14	1	12-1C12-DC6	SPDT Relay 1 form C	12A switch/6V coil	K1
15	1	13-0301	Switch	3-position DIP	S2
16	1	13-1000	Switch	Omron NO, momentary	S1
17	1	14-337	Inductor	330uH Linear	L1
18	1	20S-87C52	SMT 8-bit Microcontroller	16 MHz, PLCC44, Industrial, OTP	U7
19	1	21S-10368	SMT Crystal	10.3680 MHz, Industrial, HC49SD	Y1
20	1	21S-110592-IND	SMT Crystal	11.0592 MHz, Industrial, HC49SD	Y3
21	1	21S-16384-IND	SMT Crystal	16.3840 MHz, Industrial, HC49SD	Y2
22	1	22S-93C66	EEPROM SMT	Microchip, Industrial Temp, 8 pin SOIC	U12
23	1	23-4N25	OptoCoupler	Motorola	U1
24	1	23-4N35-DIP	OptoCoupler	6 DIP	U2
25	1	23S-75176	Transceiver	RS-485	U11
26	1	23S-DS1706P	MicroMonitor	3.3 Volt, Industrial Temp, 8-pin SOIC	U10
27	1	23S-LMV324	Quad Op-Amp	SO14	U3
28	1	23S-MSM7560	ADPCM CODEC	Single Rail, u-Law, SOP28	U4
29	1	23S-NC7S04	Single Inverter	Tiny Logic, 5 pin SOT-23	U13
30	1	23S-NC7S08	Single AND gate	Tiny Logic, 5 pin SOT-23	U9

ITEM	QTY	PART NO	DESCRIPTION	DETAIL	REFERENCE
31	1	23S-NC7S86	Single XOR gate	Tiny Logic, 5 pin SOT-23	U8
32	1	24-IRF9096	PCBA RF-9096	RF Module IRF9096DS 900MHz 96Kbps	U6
33	1	24S-AIC9001	IC, SST ASIC	AIC9001, PQFP-80	U5
34	1	31S-3904-SOT23	SMT NPN Transistor SOT-23		Q3
35	1	31S-3906-SOT23	SMT PNP Transistor SOT-23		Q5
36	1	32-2574-5	Voltage Regulator	Switching Reg. 5V	VR2
37	2	32-DF04M	Bridge Rectifier 1A, 400V		CR2,CR3
38	1	32S-MIC5205	4V Voltage Regulator	Industrial Temp, SOT23-5	VR1
39	1	34-160	Shottky Diode	1.0A 60V	CR8
40	1	34S-1N4743	Zener Diode	13V, 1W, SMT DL41	D1
41	6	34S-4148-SOT23	Switching Diode	1N4148 SOT23	CR1,CR4,CR5,CR6,CR7,CR9
42	1	34S-S1B	SMT General Purpose Diode	100V 1.0A SMA Package	CR10
43	2	36-4234	LED	Green, T 1-3/4	D2, D3
44	1	38-SID-P1100	Sidactor	Vblock = 75V min	Q2
45	1	38-SID-P3203B	Sidactor	Vblock = 225V min	Q1
46	1	41-337-25ESR	Capacitor	330uF 25V Radial Low ESR	C32
47	1	41-337-50	Capacitor	330uF 50V Radial 0.2"	C33
48	1	41-474-250V	Capacitor	470nF, 250V, Metallized Poly. Film, 0.9"	C35
49	4	41S-103-603	SMT Capacitor 0603, X7R	10nF	C3,C5,C9,C27
50	12	41S-104-603	SMT Chip Capacitor, X7R	100nF, 25V +/-10%	C1, C6, C10, C11, C12,
51	2	41S-105-1206	SMT Capacitor 1206	1uF	C2,C7
52	3	41S-105-16	SMT Capacitor, tant.	1uF 16V, -55 to +125 C, 3216 Size	C4, C22, C24
53	1	41S-106-3528	SMT Capacitor EIA std 3528	10uF 16V tant. -55 to +125 C	C28
54	1	41S-226-16-B	SMT Polarized Capacitor B type (3528)	22uF, 16V, -55 to +125 C, 10%	C21
55	3	41S-330-603	SMT Capacitor 0603, NPO	33pF, 50V	C8, C29, C30
56	1	41S-471-603	SMT Capacitor 0603, X7R	470pF, 50V	C23
57	1	51-472-0505	Resistor	4k7 1% 0.6W Metal Film	R5
58	4	51-8R2-3005	Resistor	8.2 ohm, 3 watt, High Surge	R8,R9,R10,R11
59	2	51-910-3005	Resistor	91 Ohm, 3 Watt, 1000 Volt	R42, R44
60	1	51S-000-603	SMT Resistor 0603	0 Ohm 5%, 1/16 W	R49
61	1	51S-102-603	SMT Resistor 0603	1 KOhm 5%, 1/16 W, 50V	R24
62	18	51S-103-603	SMT Resistor 0603	10K Ohm 5%, 1/16 W, 50V	R1, R3, R4, R13-R16, R18, R20, R21, R31-33, R35, R40, R43, R47, R48
63	2	51S-122-603	SMT Resistor 0603	1K2 Ohm 5%, 1/16 W, 50V	R37, R50
64	2	51S-301-603	SMT Resistor 0603	300 Ohm 5%, 1/16 W, 50V	R6, R12
65	1	51S-331-603	SMT Resistor 0603	330 Ohm 5%, 1/16 W, 50V	R23
66	3	51S-333-603	SMT Resistor 0603	33 kOhm 5%, 1/16 W, 50V	R26, R27, R28
67	2	51S-391-603	SMT Resistor 0603	390 Ohm 5%, 1/16 W, 50V	R7, R29
68	3	51S-472-603	SMT Resistor 0603	4.7 KOhm 5%, 1/16W	R19, R30, R45
69	1	51S-622-603	SMT Resistor 0603	6K2 Ohm 5%, 1/16W	R22
70	1	90-18-06-REV3	ASSEMBLY	PWA Assembly	Assembly

TelePATH
Millennium Specification
24 Dec 99

This specification is specific to the Millennium phone. See TelePATH General Specification for additional product capabilities.

General	
Digital voice coding	32 kbps ADPCM
Voice delay	<10 ms end-to-end
Signalling	T&R reversal
Data Rate	Up to 14.4 kbps (supports Bell 212A, V.32bis)
Dimensions	5.0"x6.5"x2.5" (HxWxD)
Weight	3 lb
Operating temperature	Standard 0° to 50° C Extended -40° to 85° C
Humidity	0 to 95% non-condensing
Radio	
Frequency Range	ISM, 902-928 MHz
Channels	8
RF Modulation	GMSK
Digital Modulation	Direct Sequence Digital Spread Spectrum
Bandwidth	2.3 MHz
Output power	100 mW (20 dBm +/- 2 dBm)
Receiver sensitivity	-90 dBm @ 10e-3 BER
Antenna	3 dBd omni-directional 6 dBd directional
Certification	IC RSS210 Issue 2, FCC Part 15 Subpart B and C
Phone	
Interface	600 Ω, loop-start
Ringing Amplitude	> 45 Vrms@ 1 REN
Ringing Frequency	20 Hz
Ringer Waveform	Sinusoidal
Battery Voltage	-48 Vdc
Loop Current	25 mA
Supply Voltage	10 to 36 Vdc
Supply Power	2.75 W (max)
Power Termination	Screw Terminal (2 position) + / -
Loop Termination	Screw terminal (2 position) T / R
Burst Sync Termination	Screw Terminal (4 position) M, D, /D, G
Cable Access	Through weather resistant strain relief
Antenna Termination	External reverse TNC connector
Protection	Secondary, GR-1089-CORE
Certification	IC CS03 Issue 8, FCC Part 68 Subpart D
Line	
Interface	> 600 Ω, loop-start
Ringing load	< 1 REN
Ring detect	40 - 120 Vrms
Supply Voltage	10 to 36 Vdc
Supply Power	1.75 W (max)
Power Termination	Screw Terminal (3 position) + / - / Earth ground
Loop Termination	Screw terminal (2 position) Tip / Ring
Burst Sync Termination	Screw Terminal (4 position) Master, Data, /Data, Ground
Cable Access	Through weather resistant strain relief
Antenna Termination	External reverse TNC connector
Protection	Primary, Secondary, UL 1459, CSA C22.2 No. 225
Certification	IC CS03 Issue 8, FCC Part 68 Subpart D
Craft Support	
Power On	LED
Communication Link	LED
RF Receive Signal Strength	Analog output, 0 to 1 Vdc

Critical Control

TelePATH / Millennium

USER MANUAL

Table of Contents

1)

2) OVERVIEW

Thank you for purchasing a TelePATH Millennium digital wireless link from Critical Control. Your new TelePATH Millennium is a carefully engineered, high-quality, durable product. It is designed to give you the quality and performance you expect in telecommunications equipment.

The TelePATH Millennium digital wireless link is a cost effective solution for point-to-point voice and data communication. The system provides a reliable toll-quality wireless connection between a public telephone and the wire connection back to the central office. The TelePATH Millennium may be used for both temporary and permanent installations where circumstances make conventionally wired installations impractical.

3) DESCRIPTION AND FEATURES

The TelePATH Millennium system consists of two interface enclosures, two antennas, and two coaxial cables. The two interface enclosures and antennas are installed to provide a transparent wireless link between a phone installation and the central office. The "Line Interface" connects to the central office line. The "Phone Interface" connects to the phone. The phone and line interfaces are similar in appearance but each is clearly labelled to avoid confusion.

The TelePATH Millennium has been specifically designed for use with the Nortel Millennium payphone and supports all of its functions.

The TelePATH Millennium system must be professionally installed in compliance with Industry Canada's Radio Standards Specification RSS-210, Issue 2, Rev.1.

4) INSTALLATION

A) EQUIPMENT REQUIRED

- Digital Multi-Meter
- #2 Robertson Screw Driver
- Flat Blade Jewellers Screw Driver
- Cordless Drill (Optional)
- 5/32" Drill Bit
- #2 Robertson Driver Bit (Optional)
- Static Ground Strap

A) LINE INTERFACE AND ANTENNA

I) MOUNTING INTERFACE

Identify and locate the Line Interface in an area suitable for obtaining access to dial tone and power. The location selected must also allow for connection to the antenna. Fix the Interface to the desired location using the two screws provided.

II) MOUNTING ANTENNA

Select and install the line antenna, in the desired location, with the hardware provided. For mast type installations to allow for connection of the antenna install the mast as close as possible to the Line Interface. **Masts are not provided as part of the TelePATH Millennium system.** The antenna must be installed a least two metres above grade. Where directional antennas are employed, the antennas must be installed so that the elements are vertical in orientation (**Figure 1**). For optimal system performance the antenna should be mounted as far above grade as possible.

III) ANTENNA CONNECTION

Connect the line interface to the line antenna using the coaxial cable provided. The small connector is for connection to the line Interface. The large connector is for connection to the antenna. For long-term out-door installations it recommended that the antenna connections be protected from corrosion by some suitable method (i.e. Tape Coat).

IV) TIP AND RING CONNECTION

Route the cable, providing tip and ring signalling from the central office, through the appropriate strain relief connector and terminate the tip and ring wires at the screw terminals labelled "T" and "R" respectively (**Figure 2**). Tighten the strain relief connector onto the cable finger tight. **Do not use a wrench!**

It is recommended that ground strap be used when making connections to the printed circuit boards in the interface enclosures. Failure to use a ground strap may result in permanent damage to the interface from static discharge!

V) POWER AND GROUND CONNECTIONS

Route cable providing the DC power and ground through the appropriate strain relief connector and terminate the wires at the screw terminal labelled "+" and "-". Terminate the earth ground wire at the screw terminal labelled "E" (**Figure 2**). Tighten the strain relief connector onto the cable finger tight. **Do not use a wrench!**

VI) CHANNEL SELECTION

Any of the eight channels may be selected; however, your selection must be the same for both the line and phone Interface units. Select the desired channel using the dip switches (**Figure 6**) and then momentarily press the button labelled "RESET". If interference is encountered, repeat the procedure with a different channel.

B) PHONE INTERFACE AND ANTENNA

I) MOUNTING INTERFACE

Identify and locate the Phone Interface in a se[redacted] area suitable for obtaining access to power and the [redacted]e (Booth Header, Pedestal Interface, etc.). The location selected must allow for connection to the antenna. Fix the Interface to the desired location using the two screws provided.

II) MOUNTING ANTENNA

Select and install the phone antenna in the desired location with the hardware provided. For mast type installations, install the mast as close as possible to the Phone Interface. The antenna must be installed at least two meters above grade. Where directional antennas are employed, the antennas must be installed so that the elements are vertical in orientation (See Figure 1). For optimal system performance the antenna should be mounted as far above grade as possible.

III) ANTENNA CONNECTION

Connect the Phone Interface to the antenna using the coaxial cable provided. As with the line antenna, the small connector is for connection to the phone Interface. The large connector is for connection to the antenna. For long-term out-door installations it recommended that the antenna connections be protected from corrosion by some suitable method (i.e. Tape Coat).

IV) TIP AND RING CONNECTION

Route the cable, providing tip and ring signalling to the phone, through the appropriate strain relief connector and terminate wires at the screw terminals labelled "T" and "R" (See Figure 3). Tighten the strain relief connector onto the cable finger tight. **Do not use a wrench!**

V) POWER AND GROUND CONNECTIONS

Route cable providing the DC power supply and ground through the appropriate strain relief connector and terminate the wires at the screw terminal labelled "+" and "-" (See Figure 3). The Phone Interface Unit does not require an earth ground connection. Tighten the strain relief connector onto the cable finger tight. **Do not use a wrench!**

VI) CHANNEL SELECTION

As previously done for the line interface, any of the eight channels may be selected; however, your selection must be the same for both the line and phone Interface units. Select the desired channel using the dip switches (Figure 6) and then momentarily press the button labelled "RESET". If interference is encountered, repeat the procedure with a different channel.

C) GANGED MULTIPLE LINE INSTALLATIONS

I) BURST SYNC CONNECTIONS

The TelePATH Millennium has been designed to accommodate up to eight co-located systems. In addition to the installation instructions outlined in the previous sections, when the TelePATH Millennium is employed in a multiple line installation, connections must be made in the following manner:

- Locate the terminal strip marked "G", "/D", "D", and "M" on each of the line interfaces.
- Select one of the line interfaces as the "Master" interface. The remaining interface boxes will be "Slaves" to the "Master". The master interface will co-ordinate communication between the interfaces to prevent RF interference.
- Routing them through the appropriate strain relief connectors, run a "Daisy Chain" (in series) of jumper wires from the master interface "G" terminal to each of the interface terminals marked "G" (Ground). Repeat this for both the "D Bar" and "D" terminals (See Figure 4).
- Install a jumper wire between the "G" and "M" terminals of each "Slave" interface. This indicates to the line interface that it is functioning as a "Slave" to the designated "Master" interface.
- Tighten each of the burst sync strain relief connectors onto the jumper cables finger tight.
- The burst sync connections are now complete.

I) ANTENNA SHARING

II) CHANNEL SELECTION

Where TelePATH Millennium systems are co-located, the channel selection must be set as outlined in the previous sections. In addition, the channels selected must be different for each matched set of Interface units. Assign and set each co-located system to one of the eight available channels ensuring that there is no duplication (See Figure 6).

A) CRAFT INTERFACE

- Apply power to both the Line and Phone Interface units. Verify that each green LED, marked "PWR" is illuminated.
- Verify that each LED, marked "COM", is illuminated. This LED indicates that the RF link is active between the Line and Phone interfaces.
- For line of site installations, using directional Antennas, first visually align the Antennas towards each other. To fine tune the alignment attach the leads from a digital multi-meter to the terminals marked RSSI on either the Line or Phone Interface units. Set the meter to read a range of 0 to 1 VDC. For each antenna, move the antenna until the voltage reading reaches a maximum.
- The system should now be operational. Pick up the receiver, of the phone being serviced, and verify that a call cam be completed.

18) SERVICE INFORMATION

Installation assistance may be obtained by calling (306) 382-3301.

With the exception of the five fuses, your TelePATH Millennium does not contain any field serviceable parts. Should a problem arise beyond replacement of one the fuses:

1. The defective equipment should be removed and replaced.
2. Contact Critical Control at (306) 382-3301 to obtain the required RMA (Return Materials Authorization) number which must appear on all shipping documentation.
3. Pack Product and return prepaid to:

Critical Control
Bay 6, 816 First Avenue N
Saskatoon, Saskatchewan
Canada S7K 1Y3

During the warranty period, and subject to the terms of the warranty, the equipment will be repaired and replaced at no charge.

4) SPECIFICATIONS

General

Digital voice coding	32 kbps ADPCM
Voice delay	< 10 ms end-to-end
Signalling	T&R reversal
Data Rate	Up to 14.4 kbps (supports Bell 212A, V.32bis)
Dimensions	5.0"x6.5"x2.5" (HxWxD)
Weight	3 lb
Operating temperature	Standard 0° to 50° C Extended -40° to 85° C
Humidity	0 to 95% non-condensing

Radio

Frequency Range	ISM, 902-928 MHz
Channels	8
RF Modulation	GMSK
Digital Modulation	Direct Sequence Spread Spectrum
Bandwidth	2.3 MHz
Output power	100 mW (20 dBm +/- 2 dBm)
Receiver sensitivity	-90 dBm @ 10e-3 BER
Antenna	3 dBd omni-directional 6 dBd directional
Certification	IC RSS210 Issue 2, FCC Part 15 Subpart B and C

Phone

Interface	600 Ω, loop-start
Ringing Amplitude	> 45 Vrms @ 1 REN
Ringing Frequency	20 Hz
Ringer Waveform	Sinusoidal
Battery Voltage	-48 Vdc
Loop Current	25 mA
Supply Voltage	10 to 36 Vdc
Supply Power	2.75 W (max)
Power Termination	Screw Terminal (2 position) + / -
Loop Termination	Screw terminal (2 position) T / R
Burst Sync Termination	Screw Terminal (4 position) M, D, /D, G
Cable Access	Through weather resistant strain relief
Antenna Termination	External reverse TNC connector
Protection	Secondary, GR-1089-CORE
Certification	IC CS03 Issue 8, FCC Part 68 Subpart D

Line

Interface	> 600 Ω, loop-start
Ringing load	< 1 REN
Ring detect	40 - 120 Vrms
Supply Voltage	10 to 36 Vdc
Supply Power	2.5 W (max)
Power Termination	Screw Terminal (3 position) + / - / E
Loop Termination	Screw terminal (2 position) T / R
Burst Sync Termination	Screw Terminal (4 position) M, D, /D, G
Cable Access	Through weather resistant strain relief
Antenna Termination	External reverse TNC connector
Protection	Primary, Secondary, UL 1459, CSA C22.2 No. 225
Certification	IC CS03 Issue 8, FCC Part 68 Subpart D

Craft Support

Power On	LED
Communication Link	LED
RF Receive Signal Strength	Analog output, 0 to 1 Vdc

5) LIMITED WARRANTY

The TelePATH Millennium product is warranted against manufacturing defects for one year from date of purchase. Within this period Critical Control Corp. will repair the product without charge for parts and labour. Warranty does not cover transportation costs. Nor does it cover a product subjected to misuse,

accidental damage, or unsuitable operating [redacted]tions. Except as provided herein, Critical Control [redacted] no warranties, expressed or implied, including warranties of merchantability and fitness for a particular purpose.

6) CUSTOMER INFORMATION

A) ANALOGUE DEVICE WARNINGS

This equipment complies with Part 68 of the Federal Communications Commission (FCC) rules for the United States.

A label is located on the underside of the base unit containing either the FCC registration number and Ringer Equivalence Number (REN). You must upon request, provide the following information to your local telephone company:

Facility Interface Code: NE9TM2000
Service Order Code:
USOC Jack Type:
REN:

Should you experience trouble with this telephone equipment, please contact:

Critical Control Corp.
Bay 6, 816 First Avenue N
Saskatoon, Saskatchewan
Canada S7K 1Y3

The REN is used to determine the quantity of devices which may be connected to the telephone line. Excessive RENs on the telephone line may result in the devices not ringing in response to an incoming call. In most, but not all areas, the sum of RENs should not exceed five (5.0). To be certain of the number of devices that may be connected to a line, as determined by the total RENs, contact the local telephone company.

If trouble is experienced with this equipment (TelePATH Millennium), for repair or warranty information, please contact Critical Control Corp. at (306)382-3301. If the equipment is causing harm to the telephone network, the telephone company may request that you disconnect the equipment until the problem is resolved.

This equipment cannot be used on public coin phone service provided by the telephone company. Connection to party line service is subject to state tariffs.

Your telephone company may discontinue your service if your equipment causes harm to the telephone network. They will notify you in advance of disconnection, if possible. During notification, you will be informed of your right to file a complaint to the FCC.

Occasionally, your telephone company may make changes in its facilities, equipment, operation, or procedures that could affect the operation of your equipment. If so, you will be given advance notice of the change to give you an opportunity to maintain uninterrupted service.

B) INDUSTRY CANADA WARNINGS

NOTICE:

The Industry Canada label identifies certified equipment. This certification means that the equipment meets telecommunications network protective, operational, and safety requirements as prescribed in the appropriate Terminal Equipment Technical Requirements document(s). The Department does not guarantee the equipment will operate to the user's satisfaction.

Before installing this equipment, users should ensure that it is permissible to be connected to the facilities of the local telecommunications company. The equipment must also be installed using an acceptable method of connection. The customer should be aware that compliance with the above conditions may not prevent degradation of service in some situations.

Repairs to certified equipment should be coordinated by a representative designated by the supplier. Any repairs or alterations made by the user to this equipment, or equipment malfunctions, may give the telecommunications company cause to request the user to disconnect the equipment.

Users should ensure for their own protection that the electrical ground connections of the power utility, telephone lines and internal metallic water pipe system, if present, are connected together. This precaution may be particularly important in rural areas.

Caution: Users should not attempt to make such connections themselves, but should contact the appropriate electric inspection authority, or electrician, as appropriate.

NOTICE: The Ringer Equivalence Number (REN) assigned to each terminal device provides an indication of the maximum number of terminals allowed to be connected to a telephone interface. The termination on an interface may consist of any combination of devices subject only to the requirement that the sum of the Ringer Equivalence Number of all the devices does not exceed 5.

C) RADIO INTERFERENCE STATEMENT

This device compiles with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) This device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for Class B Digital Device, pursuant to Part 15 of the FCC Rules. These limits are d3esigned to provide reasonable protection against harmful interference in a residential installation. This equipment generates and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one of more of the following measures.

- Reorient or relocate the receiving antenna
- Increase the separation between the equipment and receiver
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected
- Consult the dealer or an experience radio/TV technician for help

Any changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

(French Version)

L'utilisation de ce dispositif est autorisee seulement aux conditions suivantes: (1) Ils ne doit pas produir de brouillage, et (2) L'utilisateur du dispositif doit etre pret a accepter tout brouillage radioelectrique recu, meme si ce brouillage est susceptible de compromettre le fonctionnement du dispositif.

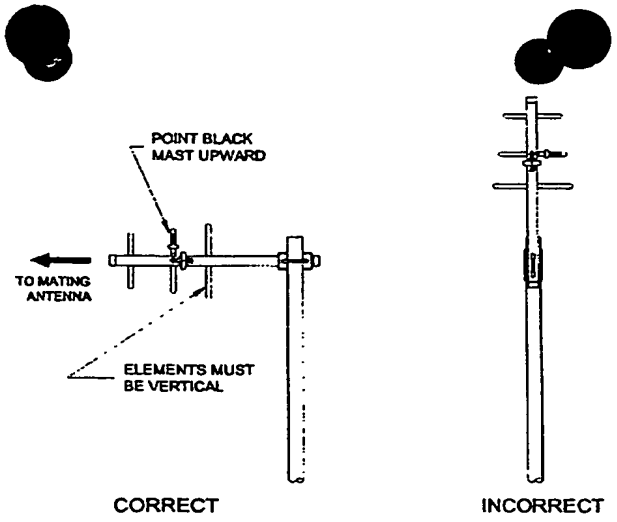


Figure 1. Antenna Orientation

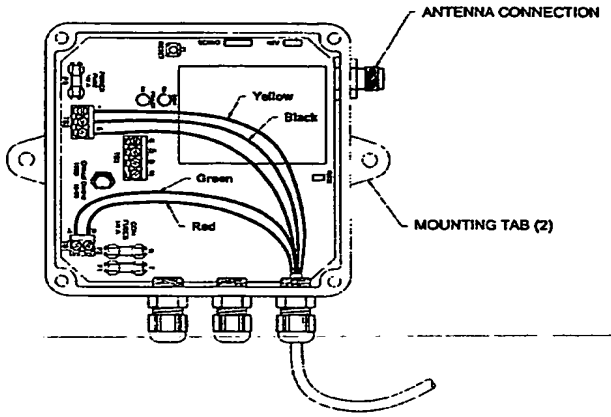


Figure 2. Line Module Wiring

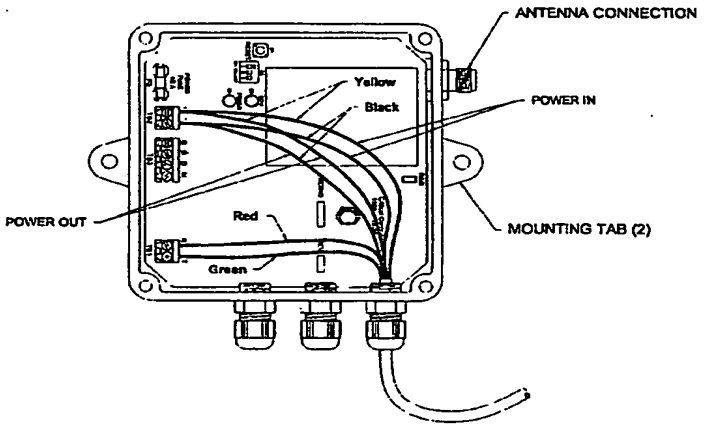


Figure 3. Phone Module Wiring

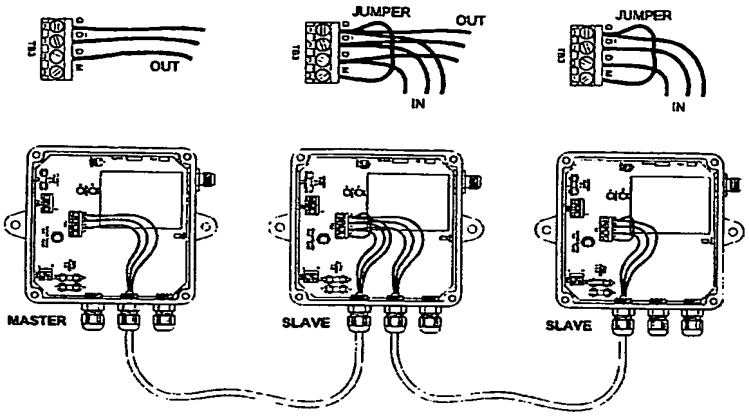


Figure 4. Master - Slave Wiring for Multiple Line Installation

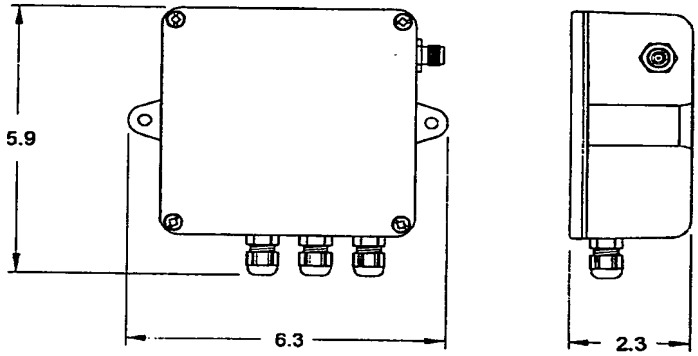


Figure 5. Enclosure Dimensions

CH	POSITION
0	ON 1 2 3
1	ON 1 2 3
2	ON 1 2 3
3	ON 1 2 3

CH	POSITION
4	ON 1 2 3
5	ON 1 2 3
6	ON 1 2 3
7	ON 1 2 3

Figure 6. Channel Selection using Dipswitch

APPENDIX A

SOURCE CODE FOR SOFTWARE IN THE PREFERRED EMBODIMENT

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\chnl\chnl.h>
#include <..\aic9001\aic9001.h>
#include <..\mb15e03\mb15e03.h>
#include <..\m93x6\m93x6.h>
#include <..\config\config.h>

// DEFINITIONS
/*****************************************************************************/
/*                                                                           */
/* Base-address definitions:                                                 */
/*                                                                           */
/* The following definitions set the base-addresses of each data             */
/* structure stored in EEPROM.                                               */
/*                                                                           */
/*****************************************************************************/
#define REF_COUNTER_BASE_ADRS   0
#define COUNTER_BASE_ADRS       ( REF_COUNTER_BASE_ADRS + sizeof(
MB15E03_RefCounterType ) )
#define PN_CODE_BASE_ADRS       ( COUNTER_BASE_ADRS + sizeof(
MB15E03_CounterType ) * MAX_CHNLS * 2 )
#define UW_BASE_ADRS            ( PN_CODE_BASE_ADRS + sizeof( Tlarge ) *
MAX_CHNLS * MAX_PN_SEQS )
#define CI_BASE_ADRS            ( UW_BASE_ADRS + sizeof( Tlarge ) * MAX_CHNLS )
#define TEST_BASE_ADRS          ( CI_BASE_ADRS + sizeof( AIC9001_CI_Type ) )

// VARIABLES
/*****************************************************************************/
/*                                                                           */
/* Data pointers:                                                            */
/*                                                                           */
/* The following pointer declarations are key to the operation of            */
/* This module. These pointers provide an efficient and elegant way          */
/* to find the address of a data structure located in a serial EEPROM        */
/* device. These pointers (once de-referenced) can be manipulated            */
/* like the structures to which they point. The C ampersand (&)              */
/* operator yields the EEPROM address of the data structure. The             */
/* address may then be used for reads or writes of the EEPROM device.        */
/*                                                                           */
/* Declaration explanation:                                                  */
/*                                                                           */
/* ConfigCounter is a pointer to an array of M15E03_CounterType sized        */
/* MAX_CHNLS x 2. ConfigCounter is located in the code segnemt (const)*/
/* and points to an array located in the idata segment. Since it             */
```

```
/* is const it occupies no RAM.  Also, as it points to idata        */
/* (zero page), it uses only one byte of memory (ROM).  This resource */
/* efficient approach uses minimum memory and processing cycles.     */
/* The pointer is initialized (at compile-time) to the base-address   */
/* of the counter array.                                              */
/*                                                                    */
/* Usage example:                                                     */
/*                                                                    */
/* void main( void )                                                  */
/* {                                                                  */
/*  Tsmall Adrs;                    // holds 8-bit address.   */
/*                                                                    */
/*  Adrs = (Tsmall)&(*ConfigCounter)[5][1]; // yields EEPROM address  */
/* }                                //  of Counter[5][1].    */
/*                                                                    */
/*                                                                    */
/**********************************************************************/
MB15E03_RefCounterType (idata * const ConfigRefCounter) = (void idata
*)REF_COUNTER_BASE_ADRS;
MB15E03_CounterType (idata * const ConfigCounter)[MAX_CHNLS][2] = (void idata
*)COUNTER_BASE_ADRS;
Tlarge (idata * const ConfigPN_Codes)[MAX_CHNLS][MAX_PN_SEQS] = (void idata
*)PN_CODE_BASE_ADRS;
Tlarge (idata * const ConfigUW)[MAX_CHNLS] = (void idata *)UW_BASE_ADRS;
AIC9001_CI_Type (idata * const ConfigCI) = (void idata *)CI_BASE_ADRS;
Tsmall (idata * const ConfigTEST) = (void idata *)TEST_BASE_ADRS;

// FUNCITONS
/**********************************************************************/
/*                                                                    */
/* ConfigRead()                                                       */
/* ------------                                                       */
/*                                                                    */
/* This function copies Len bytes from the EEPROM Src address to      */
/* the RAM Dst address.                                               */
/*                                                                    */
/* Usage:                                                             */
/*                                                                    */
/* void main( void )  // Copies from ConfigCounter[3][0] (EE)         */
/* {                  to Buf (RAM).                                   */
/*  Tsmall Buf[10];                                                   */
/*                                                                    */
/*  ConfigRead( Buf, &(*ConfigCounter)[3][0], sizeof(MB15E03_CounterType) ); */
/* }                                                                  */
/*                                                                    */
```

```
/*****************************************************************************/
void ConfigRead( void *Dst, void *Src, Tsmall Len )
{
 while( Len-- )
  *((Tsmall *)Dst)++ = M93x6_Read( (Tmedium)((Tsmall *)Src)++ );
}

/*****************************************************************************/
/*                                                                           */
/* ConfigWrite()                                                             */
/* -------------                                                             */
/*                                                                           */
/* This function copies Len bytes from the RAM Dst address to the            */
/* EEPROM Src address.                                                       */
/*                                                                           */
/* Usage:                                                                    */
/*                                                                           */
/* void main( void ) // Copies from Buf (RAM)                                */
/* {                 to ConfigCounter[3][0] (EE).                            */
/*  Tsmall Buf[10];                                                          */
/*                                                                           */
/*  ConfigWrite( &(*ConfigCounter)[3][0], Buf, sizeof(MB15E03_CounterType) );*/
/* }                                                                         */
/*                                                                           */
/*****************************************************************************/
void ConfigWrite( void *Dst, void *Src, Tsmall Len )
{
 while( Len-- )
  M93x6_Write( (Tmedium)((Tsmall *)Dst)++, *((Tsmall *)Src)++ );
}
```

```
#ifndef __AV_05__

#include <..\target\68705j1a.h>

/* Register */
/* Port Data registers */

volatile PORTA_Type          PORTA;
volatile PORTB_Type          PORTB;

/* Port Data Direction registers */
/* NOTE! These are write only */

PORTA_Type                   DDRA;
PORTB_Type                   DDRB;

/* Timer registers */

volatile TSCR_Type                TSCR;
volatile const unsigned char  TCR;

/* Interrupt control register */

volatile ISCR_Type        ISCR;

/* Port Pulldown registers */
/* NOTE! These are write only */

volatile PORTA_Type       PDRA;
volatile PORTB_Type       PDRB;

/* EPROM programming register */

volatile EPROG_Type       EPROG;

/* COP watch-dog reset register */
/* NOTE! These are write only */

volatile COPR_Type        COPR;


#endif
```

```
#ifndef CHNL_H
#define CHNL_H

/*****************************************************************************/
/*                                                                           */
/* Communications Channel Module                                             */
/* -----------------------------                                             */
/*                                                                           */
/* File: ..\chnl\chnl.h                                                      */
/* Date: 99.09.20                                                            */
/* By:   Arch                                                                */
/* Description:                                                              */
/*                                                                           */
/* The Communications Channel Module provides an interface to                */
/* the channel selection dip-switch.  A call to ChnlRead()                   */
/* returns the selected channel.                                             */
/*                                                                           */
/*****************************************************************************/

#include <..\std_inc\cc_std.h>
#include <..\mcu_def\i8xc32.h>

#define MAX_CHNLS   8
#define MAX_PN_SEQS 4

extern Tsmall Chnl;

void ChnlInit( void );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\chnl\chnl.h>

Tsmall Chnl;

/*************************************************************************/
/*                                                                       */
/* ChnlRead()                                                            */
/* ----------                                                            */
/*                                                                       */
/* This function returns the state of the channel selection switch.      */
/*                                                                       */
/*************************************************************************/
void ChnlInit( void )
{
 CHNL_SWITCH_PORT |= CHNL_SWITCH_MASK; // inputs
 Chnl = CHNL_SWITCH_PORT & CHNL_SWITCH_MASK;
}
```

```
#ifndef CC_STD_H
#define CC_STD_H

/* CC_STD.h
  Critical Control's Standard include file for all microcontroller projects
*/

#define TRUE      1
#define FALSE     0
#define ON        1
#define OFF       0
#define CLEAR     0
#define SET       1
#define ASSERT    1
#define ASSERT_N  0
#define NO        0
#define YES       1

#define Tsmall unsigned char
#define Tmedium unsigned int
#define Tlarge unsigned long
#define Tstring char*
#define Tbool unsigned char

#endif
```

```
#ifndef AIC9001_H
#define AIC9001_H

/*****************************************************************************/
/*                                                                           */
/* ALFA AIC9001 Spread Spectrum Transceiver Module                           */
/* ---------------------------------------                                   */
/*                                                                           */
/* File: ..\aic9001\aic9001.h                                                */
/* Date: 99.09.20                                                            */
/* By:   Arch                                                                */
/* Description:                                                              */
/*                                                                           */
/* The ALFA AIC9001 Spread Spectrum Transceiver Module controls              */
/* voice connections and provides a path for low-speed command               */
/* data.  The AIC9001_Init function initializes the AIC9001 device           */
/* according to the set Chnl found in the chnl module.                       */
/*                                                                           */
/* The AIC9001_Connect function allows a communications                      */
/* connection between MASTER and SLAVE devices.  The initiating              */
/* device is configured as MASTER (the other as SLAVE).                      */
/*                                                                           */
/* The AIC9001_Disconnect function closes a connection.                      */
/*                                                                           */
/* The AIC9001_RxStatusRead function delivers the status nibble              */
/* received from the low-speed data path.  The path is buffered              */
/* by a queue named RxStatus of size AIC9001_RX_Q_SIZE.                      */
/*                                                                           */
/* An outgoing status nibble may be written to AIC9001_TxStatus.             */
/* This variable (avaialbe as an extern) is periodically sent                */
/* accross the low-speed data path.  It is not buffered and is               */
/* sent once per frame (TDD cycle) while the connection is active.           */
/*                                                                           */
/* Received SNR is available in AIC9001_RxSNR.  It is updated                */
/* once per 128 bits received and is not buffered.                           */
/*                                                                           */
/*****************************************************************************/

#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\mcu_def\i8xc32.h>

typedef enum          // Configuration Information register T field
{                  // note: 3 bits wide, bit reversed
 UW_0_BIT_ERRORS = 0,  // 0 = 000  ->  000 = 0
```

```
 UW_1_BIT_ERRORS = 4,  // 1 = 001 -> 100 = 4
 UW_2_BIT_ERRORS = 2,  // 2 = 010 -> 010 = 2
 UW_3_BIT_ERRORS = 6,  // 3 = 011 -> 110 = 6
 UW_4_BIT_ERRORS = 1   // 4 = 100 -> 001 = 1
} T_Type;

typedef enum          // Configuration Information register PLSL field
{                   // note: 2 bits wide, bit reversed
 PLSL_8_SAMPLES  = 0,  // 0 = 00 -> 00 = 0
 PLSL_10_SAMPLES = 2,  // 1 = 01 -> 10 = 2
 PLSL_12_SAMPLES = 1,  // 2 = 10 -> 01 = 1
 PLSL_14_SAMPLES = 3   // 3 = 11 -> 11 = 3
} PLSL_Type;

typedef enum          // Configuration Information register CNTLR field
{
 CNTLR_10_SAMPLES,
 CNTLR_12_SAMPLES
} CNTLR_Type;

typedef enum          // Configuration Information register WSL field
{                   // note: 2 bits wide, bit reversed
 WSL_4_SAMPLES  = 0,   // 0 = 00 -> 00 = 0
 WSL_6_SAMPLES  = 2,   // 1 = 01 -> 10 = 2
 WSL_8_SAMPLES  = 1,   // 2 = 10 -> 01 = 1
 WSL_10_SAMPLES = 3    // 3 = 11 -> 11 = 3
} WSL_Type;

typedef enum
{
 SLAVE,
 MASTER
} M_SN_Type;

typedef struct        // Configuration Information register type definition
{                   // note: bit reversed
 unsigned :3;         //     (LSB of 16-bit bitfield)
 unsigned LockSMon:1;  // CI12
 unsigned T:3;        // CI9, CI10, CI11
 unsigned M_SN:1;      // CI8
 unsigned WSL:2;       // CI6, CI7
 unsigned ACC1RES:1;   // CI5
 unsigned CNTLR:1;     // CI4
 unsigned PLSL:2;      // CI2, CI3
 unsigned TestTDD:1;   // CI1
```

```
 unsigned TestMode:1;  // CI0  (MSB of 16-bit bitfield)
} AIC9001_CI_Type;

extern Tsmall AIC9001_TxStatus;
extern Tsmall AIC9001_RxSNR;

void AIC9001_Init( void );
void AIC9001_Mode( M_SN_Type Data );
void AIC9001_Connect( void );
void AIC9001_Disconnect( void );
Tbool AIC9001_RxStatusRead( Tsmall *Data );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\chnl\chnl.h>
#include <..\queue\queue.h>
//#include <..\led\led.h>
#include <..\config\config.h>
#include <..\aic9001\aic9001.h>

// TYPES
typedef enum            // SBI register addresses
{
 RX_STATUS_REG = 0x00,
 SNR_REG = RX_STATUS_REG + AIC9001_SBI_REG_OFFSET,
 PN_A_MS_REG = SNR_REG + AIC9001_SBI_REG_OFFSET,
 PN_A_LS_REG = PN_A_MS_REG + AIC9001_SBI_REG_OFFSET,
 PN_B_MS_REG = PN_A_LS_REG + AIC9001_SBI_REG_OFFSET,
 PN_B_LS_REG = PN_B_MS_REG + AIC9001_SBI_REG_OFFSET,
 PN_C_MS_REG = PN_B_LS_REG + AIC9001_SBI_REG_OFFSET,
 PN_C_LS_REG = PN_C_MS_REG + AIC9001_SBI_REG_OFFSET,
 PN_D_MS_REG = PN_C_LS_REG + AIC9001_SBI_REG_OFFSET,
 PN_D_LS_REG = PN_D_MS_REG + AIC9001_SBI_REG_OFFSET,
 UW_MS_REG = PN_D_LS_REG + AIC9001_SBI_REG_OFFSET,
 UWCI_LS_REG = UW_MS_REG + AIC9001_SBI_REG_OFFSET,
 CI_MS_REG = UWCI_LS_REG + AIC9001_SBI_REG_OFFSET,
 TX_STATUS_REG = CI_MS_REG + AIC9001_SBI_REG_OFFSET,
 TEST_REG = TX_STATUS_REG + AIC9001_SBI_REG_OFFSET,
 UNUSED_REG = TEST_REG + AIC9001_SBI_REG_OFFSET
} AIC9001_RegType;

// Data
AIC9001_CI_Type AIC9001_CI;  // Configuration information bits.
Tsmall AIC9001_TxStatus = 0xFF;// Transmit status nibble MSB justified.
Tsmall AIC9001_RxSNR;        // Received status nibble LSB justified.

// PROTOTYPES
void AIC9001_SetPN( Tsmall Chnl );
void AIC9001_SetUW( Tsmall Chnl );
void AIC9001_SetCI( void );
void AIC9001_SetTEST( void );
Tsmall AIC9001_SBI_Read( AIC9001_RegType Reg, Tsmall NumBits );
void AIC9001_SBI_Write( AIC9001_RegType Reg, Tmedium Data, Tsmall NumBits );

// FUNCTIONS

/*****************************************************************************/
```

```
/*                                                    */
/* QUEUE_CREATE( AIC9001_RxStatus, AIC9001_RX_Q_SIZE )          */
/* ---------------------------------------------------          */
/*                                                    */
/* This invocation allocates memory and declares functions for      */
/* a queue of characters AIC9001_RX_Q_SIZE bytes long that is named */
/* AIC9001_RxStatus. Functions AIC9001_RxStatusWrite and            */
/* AIC9001_RxStatusRead are declared that work directly on this     */
/* queue. The prototypes are as follows:                            */
/*                                                    */
/* Tbool AIC9001_RxStatusWrite( Tsmall Data );                      */
/* Tbool AIC9001_RxStatusRead( Tsmall *Data );                      */
/*                                                    */
/* The write function writes Data to the queue, while the read      */
/* function removes the oldest element from the queue and provides  */
/* it to the location specified by Data. Both functions return      */
/* booleans to indicate the success of the respective operation.    */
/*                                                    */
/*********************************************************************/
QUEUE_CREATE( AIC9001_RxStatus, AIC9001_RX_Q_SIZE )

/*********************************************************************/
/*                                                    */
/* AIC9001_Init()                                           */
/* --------------                                           */
/*                                                    */
/* This function initializes memory and I/O pins, it also resets    */
/* the device.                                              */
/*                                                    */
/*********************************************************************/
void AIC9001_Init( void )
{
 AIC9001_RLOCK = 1;          // input
 AIC9001_PLLSW = 1;          // input
 AIC9001_RSTF_N = 1;         // input
 AIC9001_INT_N = 1;          // input

 AIC9001_SBI_CS_N = !ASSERT_N; // deselect device.
 AIC9001_SBI_PORT |= UNUSED_REG;// set to UNUSED_REG.
 AIC9001_SBI_LATCH = CLEAR;    // idle state.
 AIC9001_SBI_DATA_OUT = SET;   // idle state.
 AIC9001_SBI_CLOCK = SET;      // idle state.

 AIC9001_Disconnect();         // make inactive.
 ConfigRead( &AIC9001_CI, &(*ConfigCI), sizeof( AIC9001_CI_Type ) );
```

```
 QueueInit( AIC9001_RxStatus ); // initialize rx status queue.
}

/*****************************************************************************/
/*                                                                           */
/* AIC9001_Mode()                                                            */
/* --------------                                                            */
/*                                                                           */
/* This function configures the device to either MASTER or SLAVE             */
/* mode as set by Data.                                                      */
/*                                                                           */
/*****************************************************************************/
void AIC9001_Mode( M_SN_Type Data )
{
 IE_EX1 = 0;                // INT1_N disabled.
 AIC9001_CI.M_SN = Data;    // make master/slave
 AIC9001_SetCI();           // set CI.
}

/*****************************************************************************/
/*                                                                           */
/* AIC9001_Connect()                                                         */
/* -----------------                                                         */
/*                                                                           */
/* This function configures the device according to the current              */
/* channel (see "..\chnl\chnl.c"), and makes it active.  It                  */
/* also enables INT1_N for low level sensitivity at low priority.            */
/*                                                                           */
/*****************************************************************************/
void AIC9001_Connect( void )
{
 IE_EX1 = 0;                // INT1_N disabled.
 AIC9001_SetPN( Chnl );     // set PN codes.
 AIC9001_SetUW( Chnl );     // set UW.
 AIC9001_SetCI();           // set CI.
 AIC9001_SetTEST();         // clear test bits.
 TCON_IT1 = 0;              // INT1_N level sensitive.
 IP_PX1 = 0;                // INT1_N low priority.
 IE_EX1 = 1;                // INT1_N enabled.
 AIC9001_CLKEN = ASSERT;
 AIC9001_RST2_N = !ASSERT_N;  // make active.
}

/*****************************************************************************/
/*                                                                           */
```

```
/* AIC9001_Disconnect()                                    */
/* --------------------                                    */
/*                                                         */
/* This function makes the device inactive (low power).  It also    */
/* disables INT1_N.                                        */
/*                                                         */
/***********************************************************************/
void AIC9001_Disconnect( void )
{
 IE_EX1 = 0;              // INT1_N disabled.
 AIC9001_RST2_N = ASSERT_N;    // make inactive.
 AIC9001_CLKEN = !ASSERT;
}

/***********************************************************************/
/*                                                         */
/* AIC9001_SBI_Read()                                      */
/* ------------------                                      */
/*                                                         */
/* This function performs a synchronous serial transfer from the    */
/* device's SBI port.  It performs all necessary hardware actions   */
/* including address selection, data clocking and latching, etc.    */
/* It reads NumBits bits from the specified SBI register (Reg)      */
/* and returns them in the LSB's of the returned byte.  Data bits   */
/* are transferred MSB first.                              */
/*                                                         */
/***********************************************************************/
Tsmall AIC9001_SBI_Read( AIC9001_RegType Reg, Tsmall NumBits )
{
 Tsmall Data = 0;

 AIC9001_SBI_PORT = ( AIC9001_SBI_PORT & ~UNUSED_REG ) | Reg;
 AIC9001_SBI_DATA_OUT = SET;
 AIC9001_SBI_CLOCK = SET;
 AIC9001_SBI_LATCH = CLEAR;
 AIC9001_SBI_CS_N = ASSERT_N;
 while( NumBits-- )
 {
  Data <<= 1;
  AIC9001_SBI_CLOCK = CLEAR;
  if( AIC9001_SBI_DATA_IN )
   Data |= 0x01;
  AIC9001_SBI_CLOCK = SET;
 }
 AIC9001_SBI_LATCH = SET;
```

```
 AIC9001_SBI_LATCH = CLEAR;
 AIC9001_SBI_PORT |= UNUSED_REG;
 AIC9001_SBI_CS_N = !ASSERT_N;
 return( Data );
}

/*************************************************************************/
/*                                                                       */
/* AIC9001_SBI_Write()                                                   */
/* -------------------                                                   */
/*                                                                       */
/* This function performs a synchronous serial transfer to the           */
/* device's SBI port.  It performs all necessary hardware actions        */
/* including address selection, data clocking and latching, etc.         */
/* It writes NumBits of the MSB's of Data to the specified SBI           */
/* register (Reg).  Data bits are transferred MSB first.                 */
/*                                                                       */
/*************************************************************************/
void AIC9001_SBI_Write( AIC9001_RegType Reg, Tmedium Data, Tsmall NumBits )
{
 AIC9001_SBI_PORT = ( AIC9001_SBI_PORT & ~UNUSED_REG ) | Reg;
 AIC9001_SBI_CLOCK = SET;
 AIC9001_SBI_LATCH = CLEAR;
 AIC9001_SBI_CS_N = ASSERT_N;
 while( NumBits-- )
 {
  AIC9001_SBI_CLOCK = CLEAR;
  if( Data & 0x8000 )
   AIC9001_SBI_DATA_OUT = SET;
  else AIC9001_SBI_DATA_OUT = CLEAR;
  AIC9001_SBI_CLOCK = SET;
  Data <<= 1;
 }
 AIC9001_SBI_DATA_OUT = SET;
 AIC9001_SBI_LATCH = SET;
 AIC9001_SBI_LATCH = CLEAR;
 AIC9001_SBI_PORT |= UNUSED_REG;
 AIC9001_SBI_CS_N = !ASSERT_N;
}

/*************************************************************************/
/*                                                                       */
/* AIC9001_SetPN()                                                       */
/* ---------------                                                       */
/*                                                                       */
```

```
/* This function sets the PN registers to the values held in       */
/* EEPROM.                                                         */
/*                                                                 */
/*******************************************************************/
void AIC9001_SetPN( Tsmall Chnl )
{
 AIC9001_RegType Reg;
 Tmedium Data;
 Tmedium *Ptr;

 Ptr = (Tmedium *)&(*ConfigPN_Codes)[Chnl][0];
 for( Reg = PN_A_MS_REG; Reg <= PN_D_LS_REG; Reg += AIC9001_SBI_REG_OFFSET
)
 {
  ConfigRead( &Data, Ptr++, sizeof( Tmedium ) );
  AIC9001_SBI_Write( Reg, Data, 16 );
 }
}

/*******************************************************************/
/*                                                                 */
/* AIC9001_SetUW()                                                 */
/* ---------------                                                 */
/*                                                                 */
/* This function sets the UW registers to the value held in        */
/* EEPROM for the current channel.  UW bits are shared with CI     */
/* bits in the device.                                             */
/*                                                                 */
/*******************************************************************/
void AIC9001_SetUW( Tsmall Chnl )
{
 Tlarge Data;

 ConfigRead( &Data, &(*ConfigUW)[Chnl], sizeof( Tlarge ) );
 Data <<= 10;
 Lo( Tmedium, Data ) |= (*(Tmedium *)&AIC9001_CI)>>6;
 AIC9001_SBI_Write( UW_MS_REG, Hi( Tmedium, Data ), 16 );
 AIC9001_SBI_Write( UWCI_LS_REG, Lo( Tmedium, Data ), 16 );
}

/*******************************************************************/
/*                                                                 */
/* AIC9001_SetCI()                                                 */
/* ---------------                                                 */
/*                                                                 */
```

```
/* This function sets the CI registers to the value held in        */
/* AIC9001_CI.  CI bits are shared with UW bits in the device.     */
/*                                                                 */
/*******************************************************************/
void AIC9001_SetCI( void )
{
 Tlarge Data;
 Tsmall UW_LoByte;

 Data = *(Tmedium *)&AIC9001_CI;
 Data <<= 10;
 ConfigRead( &UW_LoByte , &Lo( Tsmall, Lo( Tmedium, (*ConfigUW)[Chnl] ) ), sizeof(
Tsmall ) );
 Hi( Tsmall, Hi( Tmedium, Data ) ) |= UW_LoByte<<2;
 AIC9001_SBI_Write( UWCI_LS_REG, Hi( Tmedium, Data ), 16 );
 AIC9001_SBI_Write( CI_MS_REG, Lo( Tmedium, Data ), 3 );
}

/*******************************************************************/
/*                                                                 */
/* AIC9001_SetTEST()                                               */
/* -----------------                                               */
/*                                                                 */
/* This function sets the TEST register to the values held in      */
/* EEPROM.                                                         */
/*                                                                 */
/*******************************************************************/
void AIC9001_SetTEST( void )
{
 Tsmall Data;

 ConfigRead( &Data, &(*ConfigTEST), sizeof( Tsmall ) );
 AIC9001_SBI_Write( TEST_REG, Data<<3, 5 );
}

/*******************************************************************/
/*                                                                 */
/* AIC9001_ISR()                                                   */
/* -------------                                                   */
/*                                                                 */
/* This ISR services both the receive status nibble available flag */
/* and the receive signal-to-noise ratio available flag using      */
/* INT1_N.  It uses level sensitivity (active low) to accommodate  */
/* both flags with the one interrupt input.  The AIC9001_RSTF_N    */
/* helps resolve which flag is the source of the interruption.     */
```

```
/* If both flags are active, the routine services AIC9001_RSTF_N    */
/* first and then exits the ISR.  Immediately upon exit, the        */
/* ISR is vectored-to again to service the other flag.              */
/*                                                                  */
/********************************************************************/
interrupt [0x13] void AIC9001_ISR( void )
{                           // if RSTF_N is asserted
 if( AIC9001_RSTF_N == ASSERT_N ) //  write SBI's RxStatus to Q and,
 {                          //  write TxStatus to SBI.
  AIC9001_RxStatusWrite( AIC9001_SBI_Read( RX_STATUS_REG, 4 )<<4 );
  AIC9001_SBI_Write( TX_STATUS_REG, *(Tmedium *)&AIC9001_TxStatus, 8 );
 }                          // else write SBI's RxSNR to RxSNR.
 else AIC9001_RxSNR = AIC9001_SBI_Read( SNR_REG, 8 );
}
```

```
#ifndef __M68HC705J1A__
#define __M68HC705J1A__

/*****************************************************************/
/*                                                               */
/* Ports and interrupts for the 68HC705J1A microcontroller       */
/*                                                               */
/*****************************************************************/

/* Types */
typedef struct
{
 unsigned B7:1;
 unsigned B6:1;
 unsigned B5:1;
 unsigned B4:1;
 unsigned B3:1;
 unsigned B2:1;
 unsigned B1:1;
 unsigned B0:1;
 unsigned Unused1:8;
} PORTA_Type;

typedef struct
{
 const unsigned :2;
 unsigned B5:1;
 unsigned B4:1;
 unsigned B3:1;
 unsigned B2:1;
 unsigned B1:1;
 unsigned B0:1;
 unsigned Unused1:8;
} PORTB_Type;

typedef struct
{
 const unsigned TOR:1;
 const unsigned RTIF:1;
 unsigned TOIE:1;
 unsigned RTIE:1;
 unsigned TOFR:1;
 unsigned RTIFR:1;
 unsigned RT1:1;
 unsigned RT0:1;
```

```
 unsigned Unused1:8;
} TSCR_Type;

typedef struct
{
 unsigned IRQE:1;
 const unsigned :3;
 const unsigned IRQF:1;
 unsigned :1;
 unsigned IRQR:1;
 unsigned :1;
} ISCR_Type;

typedef struct
{
 unsigned :5;
 unsigned ELAT:1;
 unsigned MPGM:1;
 unsigned EPGM:1;
} EPROG_Type;
typedef struct
{
 unsigned Unused1:7;
 unsigned COPC:1;
 unsigned Unused2:8;
} COPR_Type;

#ifdef __AV_05__

/* Register */
/* Port Data registers */

static volatile PORTA_Type          PORTA @ 0x00;
static volatile PORTB_Type          PORTB @ 0x01;

/* Port Data Direction registers */
/* NOTE! These are write only */

static PORTA_Type                   DDRA  @ 0x04;
static PORTB_Type                   DDRB  @ 0x05;

/* Timer registers */

static volatile TSCR_Type         TSCR  @ 0x08;
static volatile const unsigned char  TCR  @ 0x09;
```

```
/* Interrupt control register */

static volatile ISCR_Type        ISCR @ 0x0A;

/* Port Pulldown registers */
/* NOTE! These are write only */

static volatile PORTA_Type       PDRA @ 0x10;
static volatile PORTB_Type       PDRB @ 0x11;

/* EPROM programming register */

static volatile unsigned char    EPROG @ 0x18;

/* COP watch-dog reset register */
/* NOTE! These are write only */

//static volatile unsigned char    COPRST @ 0x07F0;
static volatile COPR_Type        COPR @ 0x07F0;


/* Interrupts */

#define        TIMER_VEC 0x07F8
#define        IRQ_VEC  0x07FA
#define SWI_VEC  0x07FC
#define RESET_VEC 0x07FE

#else

/* Register */
/* Port Data registers */

extern volatile PORTA_Type       PORTA;
extern volatile PORTB_Type       PORTB;

/* Port Data Direction registers */
/* NOTE! These are write only */

extern PORTA_Type                DDRA;
extern PORTB_Type                DDRB;

/* Timer registers */

extern volatile TSCR_Type        TSCR;
```

```
extern volatile const unsigned char   TCR;

/* Interrupt control register */

extern volatile ISCR_Type        ISCR;

/* Port Pulldown registers */
/* NOTE! These are write only */

extern volatile PORTA_Type       PDRA;
extern volatile PORTB_Type       PDRB;

/* EPROM programming register */

extern volatile EPROG_Type       EPROG;

/* COP watch-dog reset register */
/* NOTE! These are write only */

extern volatile COPR_Type        COPR;


/* Interrupts */

#define        TIMER_VEC 0x07F8
#define        IRQ_VEC   0x07FA
#define SWI_VEC   0x07FC
#define RESET_VEC 0x07FE


#endif

#endif
```

```
#ifndef COMPILER_H
#define COMPILER_H

#ifdef __BC_COP8__
#define __LITTLE_BIG__
#include <..\target\cop8sax7.h>
#define Sleep() { WKPND = 0; LOPWR.HALT = 1; NOP(); NOP(); }
#define Snooze() { WKPND = 0; LOPWR.IDLE = 1; NOP(); NOP(); }
#endif

#ifdef __BC4EZWIN__
#define __LITTLE_BIG__
#define InstallVec( Vector, Function )
#define Sleep()
#define Snooze()
#define DisableInt()
#define EnableInt()
#endif

#ifdef __CVI__
#define __LITTLE_BIG__
#define InstallVec( Vector, Function ) void Function(void)
#define Sleep()
#define Snooze()
#define DisableInt()
#define EnableInt()
#define data
#define idata
#endif

#ifdef __AV6805__
#include <intrpt.h>
#define __BIG_LITTLE__
#define DisableInt() di()
#define EnableInt()  ei()
#define InstallVec( Vector, Function ) ROM_VECTOR( Vector, Function )
#define Stop()  asm( "stop" )
#endif

#ifdef __IAR8051__
#define __BIG_LITTLE__
#define InstallVec( Vector, Function ) interrupt [Vector]
#define DisableInt() IE.7=0
#define EnableInt() IE.7=1
#endif
```

```
#ifdef __GNU97R1A__
#define __LITTLE_BIG__
#define InstallVec( Vect, ISR ) extern void ISR( void ) __attribute__((interrupt_handler))
#define DisableInt() __asm__ ("orc.b #128,ccr")
#define EnableInt() __asm__ ("andc.b #63,ccr")
#endif

#ifdef __BIG_LITTLE__
#define Hi( Type, Name ) ( *( (Type *)&Name ) )
#define Lo( Type, Name ) ( *( (Type *)&Name + 1 ) )

#define HiByte( Arg ) ( *( (Tsmall *)&Arg ) )
#define LoByte( Arg ) ( *( (Tsmall *)&Arg + 1 ) )

#define HiInt( Arg ) ( *( (Tmedium *)&Arg ) )
#define LoInt( Arg ) ( *( (Tmedium *)&Arg + 1 ) )

#endif

#ifdef __LITTLE_BIG__
#define Hi( Type, Name ) ( *( (Type *)&Name + 1 ) )
#define Lo( Type, Name ) ( *( (Type *)&Name ) )

#define HiByte( Arg ) ( *( (Tsmall *)&Arg + 1 ) )
#define LoByte( Arg ) ( *( (Tsmall *)&Arg) )

#define HiInt( Arg ) ( *( (Tmedium *)&Arg + 1 ) )
#define LoInt( Arg ) ( *( (Tmedium *)&Arg) )

#endif
#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\chnl\chnl.h>
#include <..\aic9001\aic9001.h>
#include <..\mb15e03\mb15e03.h>
#include <..\m93x6\m93x6.h>
#include <..\config\config.h>

// DEFINITIONS
/*****************************************************************************/
/*                                                                           */
/* Base-address definitions:                                                 */
/*                                                                           */
/* The following definitions set the base-addresses of each data             */
/* structure stored in EEPROM.                                               */
/*                                                                           */
/*****************************************************************************/
#define REF_COUNTER_BASE_ADRS   0
#define COUNTER_BASE_ADRS       ( REF_COUNTER_BASE_ADRS + sizeof(
MB15E03_RefCounterType ) )
#define PN_CODE_BASE_ADRS       ( COUNTER_BASE_ADRS + sizeof(
MB15E03_CounterType ) * MAX_CHNLS * 2 )
#define UW_BASE_ADRS            ( PN_CODE_BASE_ADRS + sizeof( Tlarge ) *
MAX_CHNLS * MAX_PN_SEQS )
#define CI_BASE_ADRS            ( UW_BASE_ADRS + sizeof( Tlarge ) * MAX_CHNLS )
#define TEST_BASE_ADRS          ( CI_BASE_ADRS + sizeof( AIC9001_CI_Type ) )

// VARIABLES
/*****************************************************************************/
/*                                                                           */
/* Data pointers:                                                            */
/*                                                                           */
/* The following pointer declarations are key to the operation of            */
/* This module. These pointers provide an efficient and elegant way          */
/* to find the address of a data structure located in a serial EEPROM        */
/* device. These pointers (once de-referenced) can be manipulated            */
/* like the structures to which they point. The C ampersand (&)              */
/* operator yields the EEPROM address of the data structure. The             */
/* address may then be used for reads or writes of the EEPROM device.        */
/*                                                                           */
/* Declaration explanation:                                                  */
/*                                                                           */
/* ConfigCounter is a pointer to an array of M15E03_CounterType sized        */
/* MAX_CHNLS x 2. ConfigCounter is located in the code segnemt (const)*/
/* and points to an array located in the idata segment. Since it             */
```

```
/* is const it occupies no RAM.  Also, as it points to idata        */
/* (zero page), it uses only one byte of memory (ROM).  This resource */
/* efficient approach uses minimum memory and processing cycles.     */
/* The pointer is initialized (at compile-time) to the base-address   */
/* of the counter array.                                              */
/*                                                                    */
/* Usage example:                                                     */
/*                                                                    */
/* void main( void )                                                  */
/* {                                                                  */
/*  Tsmall Adrs;                    // holds 8-bit address.          */
/*                                                                    */
/*  Adrs = (Tsmall)&(*ConfigCounter)[5][1]; // yields EEPROM address  */
/* }                                // of Counter[5][1].             */
/*                                                                    */
/*                                                                    */
/**********************************************************************/
MB15E03_RefCounterType (idata * const ConfigRefCounter) = (void idata
*)REF_COUNTER_BASE_ADRS;
MB15E03_CounterType (idata * const ConfigCounter)[MAX_CHNLS][2] = (void idata
*)COUNTER_BASE_ADRS;
Tlarge (idata * const ConfigPN_Codes)[MAX_CHNLS][MAX_PN_SEQS] = (void idata
*)PN_CODE_BASE_ADRS;
Tlarge (idata * const ConfigUW)[MAX_CHNLS] = (void idata *)UW_BASE_ADRS;
AIC9001_CI_Type (idata * const ConfigCI) = (void idata *)CI_BASE_ADRS;
Tsmall (idata * const ConfigTEST) = (void idata *)TEST_BASE_ADRS;

// FUNCITONS
/**********************************************************************/
/*                                                                    */
/* ConfigRead()                                                       */
/* ------------                                                       */
/*                                                                    */
/* This function copies Len bytes from the EEPROM Src address to      */
/* the RAM Dst address.                                               */
/*                                                                    */
/* Usage:                                                             */
/*                                                                    */
/* void main( void )  // Copies from ConfigCounter[3][0] (EE)         */
/* {                  to Buf (RAM).                                   */
/*  Tsmall Buf[10];                                                   */
/*                                                                    */
/*  ConfigRead( Buf, &(*ConfigCounter)[3][0], sizeof(MB15E03_CounterType) ); */
/* }                                                                  */
/*                                                                    */
```

```
/*****************************************************************************/
void ConfigRead( void *Dst, void *Src, Tsmall Len )
{
 while( Len-- )
  *((Tsmall *)Dst)++ = M93x6_Read( (Tmedium)((Tsmall *)Src)++ );
}

/*****************************************************************************/
/*                                                                           */
/* ConfigWrite()                                                             */
/* -------------                                                             */
/*                                                                           */
/* This function copies Len bytes from the RAM Dst address to the            */
/* EEPROM Src address.                                                       */
/*                                                                           */
/* Usage:                                                                    */
/*                                                                           */
/* void main( void )  // Copies from Buf (RAM)                               */
/* {                  to ConfigCounter[3][0] (EE).                           */
/*  Tsmall Buf[10];                                                          */
/*                                                                           */
/*  ConfigWrite( &(*ConfigCounter)[3][0], Buf, sizeof(MB15E03_CounterType) );*/
/* }                                                                         */
/*                                                                           */
/*****************************************************************************/
void ConfigWrite( void *Dst, void *Src, Tsmall Len )
{
 while( Len-- )
  M93x6_Write( (Tmedium)((Tsmall *)Dst)++, *((Tsmall *)Src)++ );
}
```

```
#ifndef __DS80C320__
#define __DS80C320__

/*****************************************************************/
/*                                                               */
/* File: "..\target\ds80c320.h"                                  */
/* Date: 97.12.09                                                */
/* By:   Arch                                                    */
/* Description:                                                  */
/*                                                               */
/* Register and interrupt vector header file for the             */
/* DALLAS DS80C320 high-speed microcontroller using the          */
/* IAR ICC8051 C compiler version 4.25                           */
/*                                                               */
/*****************************************************************/

#ifdef __IAR8051__

#pragma language=extended
#define LOCATED( Adrs ) = Adrs
#define P0_0 P0.0
#define P0_1 P0.1
#define P0_2 P0.2
#define P0_3 P0.3
#define P0_4 P0.4
#define P0_5 P0.5
#define P0_6 P0.6
#define P0_7 P0.7
#define P1_0 P1.0
#define P1_1 P1.1
#define P1_2 P1.2
#define P1_3 P1.3
#define P1_4 P1.4
#define P1_5 P1.5
#define P1_6 P1.6
#define P1_7 P1.7
#define P2_0 P2.0
#define P2_1 P2.1
#define P2_2 P2.2
#define P2_3 P2.3
#define P2_4 P2.4
#define P2_5 P2.5
#define P2_6 P2.6
#define P2_7 P2.7
#define P3_0 P3.0
```

```
#define P3_1 P3.1
#define P3_2 P3.2
#define P3_3 P3.3
#define P3_4 P3.4
#define P3_5 P3.5
#define P3_6 P3.6
#define P3_7 P3.7
#define TCON_0 TCON.0
#define TCON_1 TCON.1
#define TCON_2 TCON.2
#define TCON_3 TCON.3
#define TCON_4 TCON.4
#define TCON_5 TCON.5
#define TCON_6 TCON.6
#define TCON_7 TCON.7
#define SCON0_0 SCON0.0
#define SCON0_1 SCON0.1
#define SCON0_2 SCON0.2
#define SCON0_3 SCON0.3
#define SCON0_4 SCON0.4
#define SCON0_5 SCON0.5
#define SCON0_6 SCON0.6
#define SCON0_7 SCON0.7
#define IE_0 IE.0
#define IE_1 IE.1
#define IE_2 IE.2
#define IE_3 IE.3
#define IE_4 IE.4
#define IE_5 IE.5
#define IE_6 IE.6
#define IE_7 IE.7
#define IP_0 IP.0
#define IP_1 IP.1
#define IP_2 IP.2
#define IP_3 IP.3
#define IP_4 IP.4
#define IP_5 IP.5
#define IP_6 IP.6
#define IP_7 IP.7
#define SCON1_0 SCON1.0
#define SCON1_1 SCON1.1
#define SCON1_2 SCON1.2
#define SCON1_3 SCON1.3
#define SCON1_4 SCON1.4
#define SCON1_5 SCON1.5
```

```
#define SCON1_6 SCON1.6
#define SCON1_7 SCON1.6
#define T2CON_0 T2CON.0
#define T2CON_1 T2CON.1
#define T2CON_2 T2CON.2
#define T2CON_3 T2CON.3
#define T2CON_4 T2CON.4
#define T2CON_5 T2CON.5
#define T2CON_6 T2CON.6
#define T2CON_7 T2CON.7
#define PSW_0 PSW.0
#define PSW_1 PSW.1
#define PSW_2 PSW.2
#define PSW_3 PSW.3
#define PSW_4 PSW.4
#define PSW_5 PSW.5
#define PSW_6 PSW.6
#define PSW_7 PSW.7
#define WDCON_0 WDCON.0
#define WDCON_1 WDCON.1
#define WDCON_2 WDCON.2
#define WDCON_3 WDCON.3
#define WDCON_4 WDCON.4
#define WDCON_5 WDCON.5
#define WDCON_6 WDCON.6
#define WDCON_7 WDCON.7
#define ACC_0 ACC.0
#define ACC_1 ACC.1
#define ACC_2 ACC.2
#define ACC_3 ACC.3
#define ACC_4 ACC.4
#define ACC_5 ACC.5
#define ACC_6 ACC.6
#define ACC_7 ACC.7
#define EIE_0 EIE.0
#define EIE_1 EIE.1
#define EIE_2 EIE.2
#define EIE_3 EIE.3
#define EIE_4 EIE.4
#define EIE_5 EIE.5
#define EIE_6 EIE.6
#define EIE_7 EIE.7
#define B_0 B.0
#define B_1 B.1
#define B_2 B.2
```

```
#define B_3 B.3
#define B_4 B.4
#define B_5 B.5
#define B_6 B.6
#define B_7 B.7
#define EIP_0 EIP.0
#define EIP_1 EIP.1
#define EIP_2 EIP.2
#define EIP_3 EIP.3
#define EIP_4 EIP.4
#define EIP_5 EIP.5
#define EIP_6 EIP.6
#define EIP_7 EIP.7

#else

#define LOCATED( Adrs )
#define sfr extern Tsmall
#define bit extern Tbool
#define P0_0 (*(SFR_BitType *)&P0).B0
#define P0_1 (*(SFR_BitType *)&P0).B1
#define P0_2 (*(SFR_BitType *)&P0).B2
#define P0_3 (*(SFR_BitType *)&P0).B3
#define P0_4 (*(SFR_BitType *)&P0).B4
#define P0_5 (*(SFR_BitType *)&P0).B5
#define P0_6 (*(SFR_BitType *)&P0).B6
#define P0_7 (*(SFR_BitType *)&P0).B7
#define P1_0 (*(SFR_BitType *)&P1).B0
#define P1_1 (*(SFR_BitType *)&P1).B1
#define P1_2 (*(SFR_BitType *)&P1).B2
#define P1_3 (*(SFR_BitType *)&P1).B3
#define P1_4 (*(SFR_BitType *)&P1).B4
#define P1_5 (*(SFR_BitType *)&P1).B5
#define P1_6 (*(SFR_BitType *)&P1).B6
#define P1_7 (*(SFR_BitType *)&P1).B7
#define P2_0 (*(SFR_BitType *)&P2).B0
#define P2_1 (*(SFR_BitType *)&P2).B1
#define P2_2 (*(SFR_BitType *)&P2).B2
#define P2_3 (*(SFR_BitType *)&P2).B3
#define P2_4 (*(SFR_BitType *)&P2).B4
#define P2_5 (*(SFR_BitType *)&P2).B5
#define P2_6 (*(SFR_BitType *)&P2).B6
#define P2_7 (*(SFR_BitType *)&P2).B7
#define P3_0 (*(SFR_BitType *)&P3).B0
#define P3_1 (*(SFR_BitType *)&P3).B1
```

```
#define P3_2 (*(SFR_BitType *)&P3).B2
#define P3_3 (*(SFR_BitType *)&P3).B3
#define P3_4 (*(SFR_BitType *)&P3).B4
#define P3_5 (*(SFR_BitType *)&P3).B5
#define P3_6 (*(SFR_BitType *)&P3).B6
#define P3_7 (*(SFR_BitType *)&P3).B7
#define TCON_0 (*(SFR_BitType *)&TCON).B0
#define TCON_1 (*(SFR_BitType *)&TCON).B1
#define TCON_2 (*(SFR_BitType *)&TCON).B2
#define TCON_3 (*(SFR_BitType *)&TCON).B3
#define TCON_4 (*(SFR_BitType *)&TCON).B4
#define TCON_5 (*(SFR_BitType *)&TCON).B5
#define TCON_6 (*(SFR_BitType *)&TCON).B6
#define TCON_7 (*(SFR_BitType *)&TCON).B7
#define SCON0_0 (*(SFR_BitType *)&SCON0).B0
#define SCON0_1 (*(SFR_BitType *)&SCON0).B1
#define SCON0_2 (*(SFR_BitType *)&SCON0).B2
#define SCON0_3 (*(SFR_BitType *)&SCON0).B3
#define SCON0_4 (*(SFR_BitType *)&SCON0).B4
#define SCON0_5 (*(SFR_BitType *)&SCON0).B5
#define SCON0_6 (*(SFR_BitType *)&SCON0).B6
#define SCON0_7 (*(SFR_BitType *)&SCON0).B7
#define IE_0 (*(SFR_BitType *)&IE).B0
#define IE_1 (*(SFR_BitType *)&IE).B1
#define IE_2 (*(SFR_BitType *)&IE).B2
#define IE_3 (*(SFR_BitType *)&IE).B3
#define IE_4 (*(SFR_BitType *)&IE).B4
#define IE_5 (*(SFR_BitType *)&IE).B5
#define IE_6 (*(SFR_BitType *)&IE).B6
#define IE_7 (*(SFR_BitType *)&IE).B7
#define IP_0 (*(SFR_BitType *)&IP).B0
#define IP_1 (*(SFR_BitType *)&IP).B1
#define IP_2 (*(SFR_BitType *)&IP).B2
#define IP_3 (*(SFR_BitType *)&IP).B3
#define IP_4 (*(SFR_BitType *)&IP).B4
#define IP_5 (*(SFR_BitType *)&IP).B5
#define IP_6 (*(SFR_BitType *)&IP).B6
#define IP_7 (*(SFR_BitType *)&IP).B7
#define SCON1_0 (*(SFR_BitType *)&SCON1).B0
#define SCON1_1 (*(SFR_BitType *)&SCON1).B1
#define SCON1_2 (*(SFR_BitType *)&SCON1).B2
#define SCON1_3 (*(SFR_BitType *)&SCON1).B3
#define SCON1_4 (*(SFR_BitType *)&SCON1).B4
#define SCON1_5 (*(SFR_BitType *)&SCON1).B5
#define SCON1_6 (*(SFR_BitType *)&SCON1).B6
```

```
#define SCON1_7 (*(SFR_BitType *)&SCON1).B6
#define T2CON_0 (*(SFR_BitType *)&T2CON).B0
#define T2CON_1 (*(SFR_BitType *)&T2CON).B1
#define T2CON_2 (*(SFR_BitType *)&T2CON).B2
#define T2CON_3 (*(SFR_BitType *)&T2CON).B3
#define T2CON_4 (*(SFR_BitType *)&T2CON).B4
#define T2CON_5 (*(SFR_BitType *)&T2CON).B5
#define T2CON_6 (*(SFR_BitType *)&T2CON).B6
#define T2CON_7 (*(SFR_BitType *)&T2CON).B7
#define PSW_0 (*(SFR_BitType *)&PSW).B0
#define PSW_1 (*(SFR_BitType *)&PSW).B1
#define PSW_2 (*(SFR_BitType *)&PSW).B2
#define PSW_3 (*(SFR_BitType *)&PSW).B3
#define PSW_4 (*(SFR_BitType *)&PSW).B4
#define PSW_5 (*(SFR_BitType *)&PSW).B5
#define PSW_6 (*(SFR_BitType *)&PSW).B6
#define PSW_7 (*(SFR_BitType *)&PSW).B7
#define WDCON_0 (*(SFR_BitType *)&WDCON).B0
#define WDCON_1 (*(SFR_BitType *)&WDCON).B1
#define WDCON_2 (*(SFR_BitType *)&WDCON).B2
#define WDCON_3 (*(SFR_BitType *)&WDCON).B3
#define WDCON_4 (*(SFR_BitType *)&WDCON).B4
#define WDCON_5 (*(SFR_BitType *)&WDCON).B5
#define WDCON_6 (*(SFR_BitType *)&WDCON).B6
#define WDCON_7 (*(SFR_BitType *)&WDCON).B7
#define ACC_0 (*(SFR_BitType *)&ACC).B0
#define ACC_1 (*(SFR_BitType *)&ACC).B1
#define ACC_2 (*(SFR_BitType *)&ACC).B2
#define ACC_3 (*(SFR_BitType *)&ACC).B3
#define ACC_4 (*(SFR_BitType *)&ACC).B4
#define ACC_5 (*(SFR_BitType *)&ACC).B5
#define ACC_6 (*(SFR_BitType *)&ACC).B6
#define ACC_7 (*(SFR_BitType *)&ACC).B7
#define EIE_0 (*(SFR_BitType *)&EIE).B0
#define EIE_1 (*(SFR_BitType *)&EIE).B1
#define EIE_2 (*(SFR_BitType *)&EIE).B2
#define EIE_3 (*(SFR_BitType *)&EIE).B3
#define EIE_4 (*(SFR_BitType *)&EIE).B4
#define EIE_5 (*(SFR_BitType *)&EIE).B5
#define EIE_6 (*(SFR_BitType *)&EIE).B6
#define EIE_7 (*(SFR_BitType *)&EIE).B7
#define B_0 (*(SFR_BitType *)&B).B0
#define B_1 (*(SFR_BitType *)&B).B1
#define B_2 (*(SFR_BitType *)&B).B2
#define B_3 (*(SFR_BitType *)&B).B3
```

```
#define B_4 (*(SFR_BitType *)&B).B4
#define B_5 (*(SFR_BitType *)&B).B5
#define B_6 (*(SFR_BitType *)&B).B6
#define B_7 (*(SFR_BitType *)&B).B7
#define EIP_0 (*(SFR_BitType *)&EIP).B0
#define EIP_1 (*(SFR_BitType *)&EIP).B1
#define EIP_2 (*(SFR_BitType *)&EIP).B2
#define EIP_3 (*(SFR_BitType *)&EIP).B3
#define EIP_4 (*(SFR_BitType *)&EIP).B4
#define EIP_5 (*(SFR_BitType *)&EIP).B5
#define EIP_6 (*(SFR_BitType *)&EIP).B6
#define EIP_7 (*(SFR_BitType *)&EIP).B7
typedef struct
{
 unsigned B0:1;
 unsigned B1:1;
 unsigned B2:1;
 unsigned B3:1;
 unsigned B4:1;
 unsigned B5:1;
 unsigned B6:1;
 unsigned B7:1;
 unsigned :8;
} SFR_BitType;

#endif

/*************************************************************/
/* Special function register definitions                     */
/*************************************************************/
sfr  P0      LOCATED( 0x80 ); // Port 0
sfr  SP      LOCATED( 0x81 ); // Stack pointer
sfr  DPL     LOCATED( 0x82 ); // Data pointer low 0
sfr  DPH     LOCATED( 0x83 ); // Data pointer high 0
sfr  DPL1    LOCATED( 0x84 ); // Data pointer low 1
sfr  DPH1    LOCATED( 0x85 ); // Data pointer high 1
sfr  DPS     LOCATED( 0x86 ); // Data pointer select
sfr  PCON    LOCATED( 0x87 ); // Power control
sfr  TCON    LOCATED( 0x88 ); // Timer/counter control
sfr  TMOD    LOCATED( 0x89 ); // Timer mode control
sfr  TL0     LOCATED( 0x8A ); // Timer 0 LSB
sfr  TL1     LOCATED( 0x8B ); // Timer 1 LSB
sfr  TH0     LOCATED( 0x8C ); // Timer 0 MSB
sfr  TH1     LOCATED( 0x8D ); // Timer 1 MSB
sfr  CKCON   LOCATED( 0x8E ); // Clock control
```

```
sfr  P1      LOCATED( 0x90 ); // Port 1
sfr  EXIF    LOCATED( 0x91 ); // External interrupt flag
sfr  SCON0   LOCATED( 0x98 ); // Serial port 0 control
sfr  SBUF0   LOCATED( 0x99 ); // Serial data buffer 0
sfr  P2      LOCATED( 0xA0 ); // Port 2
sfr  IE      LOCATED( 0xA8 ); // Interrupt enable
sfr  SADDR0  LOCATED( 0xA9 ); // Slave address register 0
sfr  SADDR1  LOCATED( 0xAA ); // Slave address register 1
sfr  P3      LOCATED( 0xB0 ); // Port 3
sfr  IP      LOCATED( 0xB8 ); // Interrupt priority
sfr  SADEN0  LOCATED( 0xB9 ); // Slave address mask enable register 0
sfr  SADEN1  LOCATED( 0xBA ); // Slave address mask enable register 1
sfr  SCON1   LOCATED( 0xC0 ); // Serial port 1 control
sfr  SBUF1   LOCATED( 0xC1 ); // Serial data buffer 1
sfr  STATUS  LOCATED( 0xC5 ); // Status register
sfr  TA      LOCATED( 0xC7 ); // Timed access register
sfr  T2CON   LOCATED( 0xC8 ); // Timer 2 control
sfr  T2MOD   LOCATED( 0xC9 ); // Timer 2 mode
sfr  RCAP2L  LOCATED( 0xCA ); // Timer 2 capture LSB
sfr  RCAP2H  LOCATED( 0xCB ); // Timer 2 capture MSB
sfr  TL2     LOCATED( 0xCC ); // Timer 2 LSB
sfr  TH2     LOCATED( 0xCD ); // Timer 2 MSB
sfr  PSW     LOCATED( 0xD0 ); // Program status word
sfr  WDCON   LOCATED( 0xD8 ); // Watchdog control
sfr  ACC     LOCATED( 0xE0 ); // Accumulator
sfr  EIE     LOCATED( 0xE8 ); // Extended interrupt enable
sfr  B       LOCATED( 0xF0 ); // B register
sfr  EIP     LOCATED( 0xF8 ); // Extended interrupt priority

/*************************************************************/
/* Bit addressible register definitions                      */
/*************************************************************/
// EIP
bit  PWDI    LOCATED( 0xFC );
bit  PX5     LOCATED( 0xFB );
bit  PX4     LOCATED( 0xFA );
bit  PX3     LOCATED( 0xF9 );
bit  PX2     LOCATED( 0xF8 );

// B

// EIE
bit  EWDI    LOCATED( 0xEC );
bit  EX5     LOCATED( 0xEB );
bit  EX4     LOCATED( 0xEA );
```

```
bit  EX3     LOCATED( 0xE9 );
bit  EX2     LOCATED( 0xE8 );

// ACC

// WDCON
bit  SMOD_1   LOCATED( 0xDF );
bit  POR     LOCATED( 0xDE );
bit  EPFI    LOCATED( 0xDD );
bit  PFI     LOCATED( 0xDC );
bit  WDIF    LOCATED( 0xDB );
bit  WTRF    LOCATED( 0xDA );
bit  EWT     LOCATED( 0xD9 );
bit  RWT     LOCATED( 0xD8 );

// PSW
bit  CY      LOCATED( 0xD7 );
bit  AC      LOCATED( 0xD6 );
bit  F0      LOCATED( 0xD5 );
bit  RS1     LOCATED( 0xD4 ); // register bank select bit 1
bit  RS0     LOCATED( 0xD3 ); // register bank select bit 0
bit  OV      LOCATED( 0xD2 );
bit  FL      LOCATED( 0xD1 );
bit  P       LOCATED( 0xD0 );

// T2CON
bit  TF2     LOCATED( 0xCF );
bit  EXF2    LOCATED( 0xCE );
bit  RCLK    LOCATED( 0xCD );
bit  TCLK    LOCATED( 0xCC );
bit  EXEN2   LOCATED( 0xCB );
bit  TR2     LOCATED( 0xCA );
bit  C_T2    LOCATED( 0xC9 );
bit  CP_RL2  LOCATED( 0xC8 );

// SCON1
bit  SM0_FE_1 LOCATED( 0xC7 );
bit  SM1_1   LOCATED( 0xC6 );
bit  SM2_1   LOCATED( 0xC5 );
bit  REN_1   LOCATED( 0xC4 );
bit  TB8_1   LOCATED( 0xC3 );
bit  RB8_1   LOCATED( 0xC2 );
bit  TI_1    LOCATED( 0xC1 );
bit  RI_1    LOCATED( 0xC0 );
```

```
// IP
bit  PS1     LOCATED( 0xBE );
bit  PT2     LOCATED( 0xBD );
bit  PS0     LOCATED( 0xBC );
bit  PT1     LOCATED( 0xBB );
bit  PX1     LOCATED( 0xBA );
bit  PT0     LOCATED( 0xB9 );
bit  PX0     LOCATED( 0xB8 );

// P3
bit  RD      LOCATED( 0xB7 );
bit  WR      LOCATED( 0xB6 );
bit  T1      LOCATED( 0xB5 );
bit  T0      LOCATED( 0xB4 );
bit  INT1    LOCATED( 0xB3 );
bit  INT0    LOCATED( 0xB2 );
bit  TXD0    LOCATED( 0xB1 );
bit  RXD0    LOCATED( 0xB0 );

// IE
bit  EA      LOCATED( 0xAF );
bit  ES1     LOCATED( 0xAE );
bit  ET2     LOCATED( 0xAD );
bit  ES0     LOCATED( 0xAC );
bit  ET1     LOCATED( 0xAB );
bit  EX1     LOCATED( 0xAA );
bit  ET0     LOCATED( 0xA9 );
bit  EX0     LOCATED( 0xA8 );

// P2

// SCON0
bit  SM0_FE_0 LOCATED( 0x9F );
bit  SM1_0    LOCATED( 0x9E );
bit  SM2_0    LOCATED( 0x9D );
bit  REN_0    LOCATED( 0x9C );
bit  TB8_0    LOCATED( 0x9B );
bit  RB8_0    LOCATED( 0x9A );
bit  TI_0     LOCATED( 0x99 );
bit  RI_0     LOCATED( 0x98 );

// P1
bit  T2      LOCATED( 0x97 );
bit  T2EX    LOCATED( 0x96 );
bit  RXD1    LOCATED( 0x95 );
```

```
bit  TXD1    LOCATED( 0x94 );
bit  INT2    LOCATED( 0x93 );
bit  INT3    LOCATED( 0x92 );
bit  INT4    LOCATED( 0x91 );
bit  INT5    LOCATED( 0x90 );

// TCON
bit  TF1     LOCATED( 0x8F );
bit  TR1     LOCATED( 0x8E );
bit  TF0     LOCATED( 0x8D );
bit  TR0     LOCATED( 0x8C );
bit  IE1     LOCATED( 0x8B );
bit  IT1     LOCATED( 0x8A );
bit  IE0     LOCATED( 0x89 );
bit  IT0     LOCATED( 0x88 );

// P0

/*****************************************************************/
/* Interrupt vector definitions                                  */
/*****************************************************************/
#ifdef __IAR8051__
interrupt [0x33] void PFI_int (void);  // Power-fail indicator
interrupt [0x03] void IE0_int (void);  // External interrupt 0
interrupt [0x0B] void TF0_int (void);  // Timer 0 overflow
interrupt [0x13] void IE1_int (void);  // External interrupt 1
interrupt [0x1B] void TF1_int (void);  // Timer 1 overflow
interrupt [0x23] void SCON0_int (void); // Serial port 0
interrupt [0x2B] void TF2_int (void);  // Timer 2 overflow
interrupt [0x3B] void SCON1_int (void); // Serial port 1
interrupt [0x43] void IE2_int (void);  // External interrupt 2
interrupt [0x4B] void IE3_int (void);  // External interrupt 3
interrupt [0x53] void IE4_int (void);  // External interrupt 4
interrupt [0x5B] void IE5_int (void);  // External interrupt 5
interrupt [0x63] void WDIF_int (void); // Watchdog interrupt
#endif
#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\mcu_def\ds80c320.h>
#ifndef __IAR8051__

/*****************************************************************/
/* Special function register definitions                         */
/*****************************************************************/
Tsmall  P0      LOCATED( 0x80 );  // Port 0
Tsmall  SP      LOCATED( 0x81 );  // Stack pointer
Tsmall  DPL     LOCATED( 0x82 );  // Data pointer low 0
Tsmall  DPH     LOCATED( 0x83 );  // Data pointer high 0
Tsmall  DPL1    LOCATED( 0x84 );  // Data pointer low 1
Tsmall  DPH1    LOCATED( 0x85 );  // Data pointer high 1
Tsmall  DPS     LOCATED( 0x86 );  // Data pointer select
Tsmall  PCON    LOCATED( 0x87 );  // Power control
Tsmall  TCON    LOCATED( 0x88 );  // Timer/counter control
Tsmall  TMOD    LOCATED( 0x89 );  // Timer mode control
Tsmall  TL0     LOCATED( 0x8A );  // Timer 0 LSB
Tsmall  TL1     LOCATED( 0x8B );  // Timer 1 LSB
Tsmall  TH0     LOCATED( 0x8C );  // Timer 0 MSB
Tsmall  TH1     LOCATED( 0x8D );  // Timer 1 MSB
Tsmall  CKCON   LOCATED( 0x8E );  // Clock control
Tsmall  P1      LOCATED( 0x90 );  // Port 1
Tsmall  EXIF    LOCATED( 0x91 );  // External interrupt flag
Tsmall  SCON0   LOCATED( 0x98 );  // Serial port 0 control
Tsmall  SBUF0   LOCATED( 0x99 );  // Serial data buffer 0
Tsmall  P2      LOCATED( 0xA0 );  // Port 2
Tsmall  IE      LOCATED( 0xA8 );  // Interrupt enable
Tsmall  SADDR0  LOCATED( 0xA9 );  // Slave address register 0
Tsmall  SADDR1  LOCATED( 0xAA );  // Slave address register 1
Tsmall  P3      LOCATED( 0xB0 );  // Port 3
Tsmall  IP      LOCATED( 0xB8 );  // Interrupt priority
Tsmall  SADEN0  LOCATED( 0xB9 );  // Slave address mask enable register 0
Tsmall  SADEN1  LOCATED( 0xBA );  // Slave address mask enable register 1
Tsmall  SCON1   LOCATED( 0xC0 );  // Serial port 1 control
Tsmall  SBUF1   LOCATED( 0xC1 );  // Serial data buffer 1
Tsmall  STATUS  LOCATED( 0xC5 );  // Status register
Tsmall  TA      LOCATED( 0xC7 );  // Timed access register
Tsmall  T2CON   LOCATED( 0xC8 );  // Timer 2 control
Tsmall  T2MOD   LOCATED( 0xC9 );  // Timer 2 mode
Tsmall  RCAP2L  LOCATED( 0xCA );  // Timer 2 capture LSB
Tsmall  RCAP2H  LOCATED( 0xCB );  // Timer 2 capture MSB
Tsmall  TL2     LOCATED( 0xCC );  // Timer 2 LSB
Tsmall  TH2     LOCATED( 0xCD );  // Timer 2 MSB
Tsmall  PSW     LOCATED( 0xD0 );  // Program status word
```

```
Tsmall  WDCON    LOCATED( 0xD8 ); // Watchdog control
Tsmall  ACC      LOCATED( 0xE0 ); // Accumulator
Tsmall  EIE      LOCATED( 0xE8 ); // Extended interrupt enable
Tsmall  B        LOCATED( 0xF0 ); // B register
Tsmall  EIP      LOCATED( 0xF8 ); // Extended interrupt priority

/*****************************************************************/
/* Bit addressible register definitions                          */
/*****************************************************************/
// EIP
Tbool  PWDI     LOCATED( 0xFC );
Tbool  PX5      LOCATED( 0xFB );
Tbool  PX4      LOCATED( 0xFA );
Tbool  PX3      LOCATED( 0xF9 );
Tbool  PX2      LOCATED( 0xF8 );

// B

// EIE
Tbool  EWDI     LOCATED( 0xEC );
Tbool  EX5      LOCATED( 0xEB );
Tbool  EX4      LOCATED( 0xEA );
Tbool  EX3      LOCATED( 0xE9 );
Tbool  EX2      LOCATED( 0xE8 );

// ACC

// WDCON
Tbool  SMOD_1   LOCATED( 0xDF );
Tbool  POR      LOCATED( 0xDE );
Tbool  EPFI     LOCATED( 0xDD );
Tbool  PFI      LOCATED( 0xDC );
Tbool  WDIF     LOCATED( 0xDB );
Tbool  WTRF     LOCATED( 0xDA );
Tbool  EWT      LOCATED( 0xD9 );
Tbool  RWT      LOCATED( 0xD8 );

// PSW
Tbool  CY       LOCATED( 0xD7 );
Tbool  AC       LOCATED( 0xD6 );
Tbool  F0       LOCATED( 0xD5 );
Tbool  RS1      LOCATED( 0xD4 ); // register bank select Tbool 1
Tbool  RS0      LOCATED( 0xD3 ); // register bank select Tbool 0
Tbool  OV       LOCATED( 0xD2 );
Tbool  FL       LOCATED( 0xD1 );
```

```
Tbool  P       LOCATED( 0xD0 );

// T2CON
Tbool  TF2     LOCATED( 0xCF );
Tbool  EXF2    LOCATED( 0xCE );
Tbool  RCLK    LOCATED( 0xCD );
Tbool  TCLK    LOCATED( 0xCC );
Tbool  EXEN2   LOCATED( 0xCB );
Tbool  TR2     LOCATED( 0xCA );
Tbool  C_T2    LOCATED( 0xC9 );
Tbool  CP_RL2  LOCATED( 0xC8 );

// SCON1
Tbool  SM0_FE_1 LOCATED( 0xC7 );
Tbool  SM1_1   LOCATED( 0xC6 );
Tbool  SM2_1   LOCATED( 0xC5 );
Tbool  REN_1   LOCATED( 0xC4 );
Tbool  TB8_1   LOCATED( 0xC3 );
Tbool  RB8_1   LOCATED( 0xC2 );
Tbool  TI_1    LOCATED( 0xC1 );
Tbool  RI_1    LOCATED( 0xC0 );

// IP
Tbool  PS1     LOCATED( 0xBE );
Tbool  PT2     LOCATED( 0xBD );
Tbool  PS0     LOCATED( 0xBC );
Tbool  PT1     LOCATED( 0xBB );
Tbool  PX1     LOCATED( 0xBA );
Tbool  PT0     LOCATED( 0xB9 );
Tbool  PX0     LOCATED( 0xB8 );

// P3
Tbool  RD      LOCATED( 0xB7 );
Tbool  WR      LOCATED( 0xB6 );
Tbool  T1      LOCATED( 0xB5 );
Tbool  T0      LOCATED( 0xB4 );
Tbool  INT1    LOCATED( 0xB3 );
Tbool  INT0    LOCATED( 0xB2 );
Tbool  TXD0    LOCATED( 0xB1 );
Tbool  RXD0    LOCATED( 0xB0 );

// IE
Tbool  EA      LOCATED( 0xAF );
Tbool  ES1     LOCATED( 0xAE );
Tbool  ET2     LOCATED( 0xAD );
```

```
Tbool  ES0     LOCATED( 0xAC );
Tbool  ET1     LOCATED( 0xAB );
Tbool  EX1     LOCATED( 0xAA );
Tbool  ET0     LOCATED( 0xA9 );
Tbool  EX0     LOCATED( 0xA8 );

// P2

// SCON0
Tbool  SM0_FE_0 LOCATED( 0x9F );
Tbool  SM1_0    LOCATED( 0x9E );
Tbool  SM2_0    LOCATED( 0x9D );
Tbool  REN_0    LOCATED( 0x9C );
Tbool  TB8_0    LOCATED( 0x9B );
Tbool  RB8_0    LOCATED( 0x9A );
Tbool  TI_0     LOCATED( 0x99 );
Tbool  RI_0     LOCATED( 0x98 );

// P1
Tbool  T2      LOCATED( 0x97 );
Tbool  T2EX    LOCATED( 0x96 );
Tbool  RXD1    LOCATED( 0x95 );
Tbool  TXD1    LOCATED( 0x94 );
Tbool  INT2    LOCATED( 0x93 );
Tbool  INT3    LOCATED( 0x92 );
Tbool  INT4    LOCATED( 0x91 );
Tbool  INT5    LOCATED( 0x90 );

// TCON
Tbool  TF1     LOCATED( 0x8F );
Tbool  TR1     LOCATED( 0x8E );
Tbool  TF0     LOCATED( 0x8D );
Tbool  TR0     LOCATED( 0x8C );
Tbool  IE1     LOCATED( 0x8B );
Tbool  IT1     LOCATED( 0x8A );
Tbool  IE0     LOCATED( 0x89 );
Tbool  IT0     LOCATED( 0x88 );

// P0

#endif
```

```
#ifndef DS1706_H
#define DS1706_H

/*************************************************************************/
/*                                                                       */
/* The DS1706 Module                                                     */
/* -----------------                                                     */
/*                                                                       */
/* File: ..\ds1706\ds1706.h                                              */
/* Date: 99.09.20                                                        */
/* By:   Arch                                                            */
/* Description:                                                          */
/*                                                                       */
/* The DS1706 Module provides an interface to the DS1706                 */
/* CPU-supervisory circuit.                                              */
/*                                                                       */
/* A call to WatchDogReset resets the watch-dog function.                */
/*                                                                       */
/*************************************************************************/

#include <..\mcu_def\i8xc32.h>

void DS1706_Reset( void );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\ds1706\ds1706.h>

/*************************************************************************/
/*                                                                       */
/* DS1706_Reset()                                                        */
/* --------------                                                        */
/*                                                                       */
/* This function resets (strobes) the DS1706 watch-dog CPU-              */
/* supervisor circuit to prevent CPU reset.                              */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
void DS1706_Reset( void )
{
 DS1706_STROBE_N = ASSERT_N;
 DS1706_STROBE_N = !ASSERT_N;
}
```

```
;------------------------------------------------------;
;                                                      ;
;                  CSTARTUP.S03                        ;
;                                                      ;
; This module contains the entire code executed before the C   ;
; "main" function is called. The code can be tailored to suit  ;
; special hardware needs. The code is designed to run on any   ;
; processor based on the 8051 architecture.            ;
;                                                      ;
; Version: 5.20 [IMAF 960603]                          ;
;
; Revision control system
;     $Revision: 1.1 $
;
;------------------------------------------------------;
  NAME  CSTARTUP
  PUBLIC init_C
  $DEFFN init_C(0400H,0,0,0,0,0,0,0)

$defmodel.inc                   ; Defines memory model

  EXTERN ?C_EXIT                ; Where to go when program is done
  EXTERN _R                   ; Register bank (0, 8, 16 or 24)
  EXTERN main                   ; First C function usually
  $DEFFN  main(32768,0,0,0)
  EXTERN  __low_level_init      ; Setup low level things
  $DEFFN  __low_level_init(0,0,0,0)
  EXTERN exit
  $DEFFN  exit(0,0,0,0)
  IF  banked_mode
  EXTERN  ?X_CALL_L18
    ENDIF

  RSEG  B_UDATA
  RSEG  B_CDATA
  RSEG  B_IDATA
  RSEG  D_UDATA
  RSEG  D_CDATA
  RSEG  D_IDATA
  RSEG  I_UDATA
  RSEG  I_CDATA
  RSEG  I_IDATA
  RSEG  P_UDATA
  RSEG  P_CDATA
  RSEG  P_IDATA
```

```
    IF    lcall_mode
    RSEG  X_UDATA
    RSEG  X_CDATA
    RSEG  X_IDATA
    RSEG  ECSTR
    RSEG  CCSTR
    ENDIF

;-------------------------------------------------------;
; The C stack segment. Should be mapped into internal data RAM ;
;-------------------------------------------------------;
; The C stack is used for LCALL's and temporary storage for   ;
; code generator help-routines (math etc). The stack will be  ;
; located after all other internal RAM variables if the stan-  ;
; dard linking procedure is followed. Note that C interrupt   ;
; routines can double stack size demands.                     ;
;-------------------------------------------------------;

    RSEG  CSTACK
stack_begin:
    DS    0                 ; Increase if needed

    COMMON INTVEC           ; Should be at location zero

;-------------------------------------------------------;
; C interrupt routines with defined [vectors] will reserve    ;
; space in this area. So will handlers written in assembler if ;
; they follow the recommended format.                         ;
;-------------------------------------------------------;

startup:
    IF    lcall_mode
    LJMP  init_C
    ELSE
    AJMP  init_C
    ENDIF


    RSEG  RCODE             ; Should be loaded after INTVEC
init_C:
    MOV   SP,#stack_begin - 1   ; From low to high addresses
;   MOV   DPTR,#SFB (P_IDATA)   ; Initialize high byte of PDATA.
;   MOV   P2,DPH                ;

;-------------------------------------------------------
```

```
; Call __low_level_init to perform initialization before initializing
; segments and calling main. If the function returns 0 no segment
; initialization should take place.
;
; Link with your own version of __low_level_init to override the
; default action: to do nothing but return 1.
;-------------------------------------------------------------

  IF   banked_mode
  MOV   A,#$BYTE3 __low_level_init
  MOV   DPTR,# LWRD(__low_level_init)
  LCALL   ?X_CALL_L18      ; main()
  ELSE
  IF   lcall_mode
  LCALL   __low_level_init
  ELSE
  ACALL   __low_level_init
  ENDIF
  ENDIF

    MOV    A,R4
    ORL    A,R5
    JZ     skip_init


;-------------------------------------------------------------;
;          C variable initialization section                  ;
;          ================================                   ;
;                                                             ;
; If there is no demand that global/static C variables should ;
; have a defined value at startup (required by ANSI), the     ;
; following section can be removed to conserve code memory    ;
; size.  Note that this part calls functions in the end of    ;
; this file, that also can be removed if initialized values are ;
; not needed.                                                 ;
;                                                             ;
; Systems controlled by a watch-dog may require additional    ;
; code insertions as the initialization can take several      ;
; milliseconds (if there are many variables) to complete.     ;
; These parts are marked with   *** WDG ***                   ;
;-------------------------------------------------------------;

;=============================================================================;
; Zero out sections containing variables without explicit     ;
; initializers like in:                                       ;
```

```
;                                                    ;
;    int i;                                          ;
;    xdata double d[10];                             ;
;====================================================;

    MOV    R0,#SFE(I_UDATA)-1
    SJMP   CLEAR_IDATA_2
CLEAR_IDATA:
    MOV    @R0,#0
    DEC    R0
CLEAR_IDATA_2:
    CJNE   R0,#SFB(I_UDATA)-1,CLEAR_IDATA

    MOV    R0,#SFE(D_UDATA)-1
    SJMP   CLEAR_DDATA_2
CLEAR_DDATA:
    MOV    @R0,#0
    DEC    R0
CLEAR_DDATA_2:
    CJNE   R0,#SFB(D_UDATA)-1,CLEAR_DDATA

    MOV    R0,#SFE(B_UDATA)-1
    SJMP   CLEAR_BDATA_2
CLEAR_BDATA:
    MOV    @R0,#0
    DEC    R0
CLEAR_BDATA_2:
    CJNE   R0,#SFB(B_UDATA)-1,CLEAR_BDATA

    IF     lcall_mode
    MOV    DPTR,#SFE (X_UDATA)          ; XDATA
    MOV    R6,DPH
    MOV    R7,DPL
    MOV    DPTR,#SFB (X_UDATA)
NEXT_X_UDATA:
    LCALL  COMP_R67_DPTR
;   JZ     INIT_P_UDATA
    JZ     INIT_VARS
    CLR    A
    MOVX   @DPTR,A
    INC    DPTR
    SJMP   NEXT_X_UDATA

;INIT_P_UDATA:
;   MOV    DPTR,#SFB (P_UDATA)
```

```
;  MOV    R0,DPL
;  MOV    DPTR,#SFE (P_UDATA)
;  MOV    R1,DPL
;NEXT_P_UDATA:
;  MOV    A,R0
;  CLR    C
;  SUBB   A,R1
;  JNC    INIT_VARS
;  MOV    @R0,#0
;  INC    R0
;  SJMP   NEXT_P_UDATA
   ENDIF

;=============================================================;
; Copy initializers into the proper memory segments for decla- ;
; rations like:                                               ;
;                                                             ;
;     int i = 7;                                              ;
;     idata char *cp = "STRING";                              ;
;=============================================================;

INIT_VARS:
   MOV    DPTR,#BDATA_TABLE          ; BDATA
   IF     lcall_mode
   LCALL  DI_INIT
   ELSE
   ACALL  DI_INIT
   ENDIF
   MOV    DPTR,#DATA_TABLE           ; DATA
   IF     lcall_mode
   LCALL  DI_INIT
   ELSE
   ACALL  DI_INIT
   ENDIF
   MOV    DPTR,#IDATA_TABLE          ; IDATA
   IF     lcall_mode
   LCALL  DI_INIT
   ELSE
   ACALL  DI_INIT
   ENDIF
;  MOV    DPTR,#PDATA_TABLE          ; PDATA
;  IF     lcall_mode
;  LCALL  X_INIT
;  ENDIF
   IF     lcall_mode
```

```
  MOV   DPTR,#XDATA_TABLE          ; XDATA
  LCALL  X_INIT
  MOV   DPTR,#YDATA_TABLE          ; For the -y compiler option
  LCALL  X_INIT
  ENDIF

skip_init:

;-------------------------------------------------------;
;       C variable initialization section end           ;
;-------------------------------------------------------;

;-------------------------------------------------------;
; Activate the (at link-time) selected register bank.   ;
;-------------------------------------------------------;

  MOV   PSW,#_R

;-------------------------------------------------------;
; If hardware must be initiated from assembly or if interrupts ;
; should be on when reaching main, this is the place to insert ;
; such code.                                            ;
;-------------------------------------------------------;

  IF    banked_mode

  MOV   A,#$BYTE3 main
  MOV   DPTR,#LWRD(main)
  LCALL  ?X_CALL_L18          ; main()

  ELSE

  IF    lcall_mode
  LCALL  main                 ; main()
  ELSE
  ACALL  main                 ; main()
  ENDIF
  ENDIF

;-------------------------------------------------------;
; Now when we are ready with our C program (usually 8051 C  ;
; programs are continuous) we must perform a system-dependent ;
; action.  In this simple case we just stop.            ;
;-------------------------------------------------------;
; DO NOT CHANGE NEXT LINE OF CSTARTUP IF YOU WANT TO RUN YOUR ;
```

```
; SOFTWARE WITH THE AID OF THE C-SPY HLL DEBUGGER. IT CAN     ;
; THOUGH BE REMOVED IF YOUR PROGRAM IS CONTINUOUS (NO EXIT).  ;
; If it is removed the EXTERN ?C_EXIT line should also be re- ;
; moved to avoid linking of the "exit" module                 ;
;-------------------------------------------------------------;
  IF    lcall_mode
  LJMP  exit
  ELSE
  AJMP  exit
  ENDIF

;-------------------------------------------------------------;
; Last part of the C variable initializer code.               ;
;-------------------------------------------------------------;
  IF    lcall_mode
COMP_R67_DPTR:
;=============================;
;     *** WDG ***             ;
;                             ;
; ACC, B, R0-R3 may be used ;
;=============================;
  MOV   A,R7
  XRL   A,DPL
  JNZ   NOT_EQUAL
  MOV   A,R6
  XRL   A,DPH
NOT_EQUAL:
  RET
  ENDIF

DI_INIT:
  CLR   A
  MOVC  A,@A+DPTR
  MOV   R0,A
  MOV   A,#1
  MOVC  A,@A+DPTR
  MOV   R1,A
  MOV   A,#2
  MOVC  A,@A+DPTR
  MOV   R6,A
  MOV   A,#3
  MOVC  A,@A+DPTR
  MOV   DPL,A
  MOV   DPH,R6
MORE_DI_COPY:
```

```
;=========================================;
;     *** WDG ***         ;
;                         ;
; ACC, B, R2-R6 may be used ;
;=========================================;
  MOV    A,R0
  XRL    A,R1
  JNZ    $+3
  RET
  CLR    A
  MOVC   A,@A+DPTR
  MOV    @R0,A
  INC    DPTR
  INC    R0
  SJMP   MORE_DI_COPY

  IF     lcall_mode
X_INIT:
  CLR    A
  MOVC   A,@A+DPTR
  MOV    R4,A
  MOV    A,#1
  MOVC   A,@A+DPTR
  MOV    R5,A
  MOV    A,#2
  MOVC   A,@A+DPTR
  MOV    R6,A
  MOV    A,#3
  MOVC   A,@A+DPTR
  MOV    R7,A
  MOV    A,#4
  MOVC   A,@A+DPTR
  MOV    R0,A
  MOV    A,#5
  MOVC   A,@A+DPTR
  MOV    DPL,A
  MOV    DPH,R0
MORE_X_COPY:
  LCALL  COMP_R67_DPTR
  JNZ    $+3
  RET
  CLR    A
  MOVC   A,@A+DPTR
  INC    DPTR
  MOV    R0,DPH
```

```
    MOV   R1,DPL
    MOV   DPH,R4
    MOV   DPL,R5
    MOVX  @DPTR,A
    INC   DPTR
    MOV   R4,DPH
    MOV   R5,DPL
    MOV   DPH,R0
    MOV   DPL,R1
    SJMP  MORE_X_COPY
    ENDIF

  BDATA_TABLE:
    DB    SFB (B_IDATA)
    DB    SFE (B_IDATA)
    DW    SFB (B_CDATA)
  DATA_TABLE:
    DB    SFB (D_IDATA)
    DB    SFE (D_IDATA)
    DW    SFB (D_CDATA)
  IDATA_TABLE:
    DB    SFB (I_IDATA)
    DB    SFE (I_IDATA)
    DW    SFB (I_CDATA)
  PDATA_TABLE:
    DW    SFB (P_IDATA)
    DW    SFE (P_CDATA)
    DW    SFB (P_CDATA)

    IF    lcall_mode
  XDATA_TABLE:
    DW    SFB (X_IDATA)
    DW    SFE (X_CDATA)
    DW    SFB (X_CDATA)
  YDATA_TABLE:
    DW    SFB (ECSTR)
    DW    SFE (CCSTR)
    DW    SFB (CCSTR)
    ENDIF

    ENDMOD  startup

;-------------------------------------------------------------;
; Function/module: exit (int code)                           ;
;                                        ;
```

```
; When C-SPY is used this code will automatically be replaced  ;
; by a 'debug' version of exit().                               ;
;-----------------------------------------------------------;
  MODULE exit

  PUBLIC exit
  $DEFFN exit(0,0,0,0,0,0,0,0)
  PUBLIC ?C_EXIT

  RSEG   RCODE

?C_EXIT:
exit:
;-----------------------------------------------------------;
; The next line could be replaced by user defined code.     ;
;-----------------------------------------------------------;
  SJMP   $          ; Forever...

  ENDMOD

;-------------------------------------------------------------
;
; The only action of this default version of __low_level_init is to
; return 1. By doing so it signals that normal initialization of data
; segments should be done.
;
; A customized version of __low_level_init may be created in order to
; perform initialization before initializing segments and calling main
; and/or to skip initialization of data segments under certain
; circumstances.
; For further details see sample file lowinit.c
;
;-------------------------------------------------------------


  MODULE   lowinit
  PUBLIC   __low_level_init
  $DEFFN   __low_level_init(0,0,0,0,32768,0,0,0)
  RSEG   CODE

__low_level_init:
  MOV   R4,#1           ; By returning 1 this function
  MOV   R5,#0         ; indicates that the normal
          ; initialization should take place
```

```
IF  banked_mode

EXTERN  ?X_RET_L18
LJMP  ?X_RET_L18

ELSE

RET

ENDIF

END
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\config\config.h>
#include <..\aic9001\aic9001.h>
#include <..\mb15e03\mb15e03.h>
#include <..\m93x6\m93x6.h>
#include <..\pkt\pkt.h>

// CONSTANT DATA
const MB15E03_RefCounterType RefCounter =
{//SW FC LDS CS              // Reference counter.
 { 1, 1, 0, 0 }, 64
};

const MB15E03_CounterType Counter[MAX_CHNLS][2] =
{                           // Programmable counter.
 {{ 58,  7 }, { 55, 27 }},   // * Channel 0
 {{ 58, 16 }, { 55, 36 }},   //     .
 {{ 58, 25 }, { 55, 45 }},   //     .
 {{ 58, 34 }, { 55, 54 }},   //     .
 {{ 58, 44 }, { 56,  0 }},   //     .
 {{ 58, 54 }, { 56, 10 }},   //     .
 {{ 59,  0 }, { 56, 20 }},   //     .
 {{ 59,  9 }, { 56, 29 }}    // * Channel 7
};

const Tlarge PN_Codes[MAX_CHNLS][MAX_PN_SEQS] =
{
 { 0x83E80A70l, 0xF33C8196l, 0x129596FAl, 0x087A249Al },
 { 0x054C5513l, 0x8EA24F87l, 0xD435C92Bl, 0x4F5168B5l },
 { 0x50BAA739l, 0xBB83321Bl, 0x42A759ABl, 0x8CE2E3C3l },
 { 0x78D465D2l, 0xAC5AD2B2l, 0xC4823B50l, 0x655D9D14l },
 { 0xE9ADEBD8l, 0xC61E7A8Al, 0x4DF29B0Cl, 0x1368D79Al },
 { 0x8C3CF515l, 0xA153ACD5l, 0x77066437l, 0xC18A55EDl },
 { 0x68CAA59El, 0xDC8F4654l, 0xF1A8CBA4l, 0xF4405D7Al },
 { 0xD6AD88D6l, 0x5598D6A5l, 0x96CAF149l, 0x673968691 }
};

const Tlarge UW[MAX_CHNLS] =
{
 0x18011Al,
 0x047123l,
 0x185231l,
 0x1655E6l,
 0x39032Dl,
```

```
 0x175D25l,
 0x16D645l,
 0x0BAD5Bl
};

const AIC9001_CI_Type CI =
{
 TRUE,
 UW_3_BIT_ERRORS,
 SLAVE,
 WSL_8_SAMPLES,
 FALSE,
 CNTLR_10_SAMPLES,
 PLSL_12_SAMPLES,
 FALSE,
 FALSE
};

const Tsmall TEST = 0;

void main( void )
{
 Tsmall Row, Column;
 Tmedium Adrs;
 MB15E03_RefCounterType TempRefCounter;
 MB15E03_CounterType TempCounter;
 Tlarge TempPN_Codes;
 Tlarge TempUW;
 AIC9001_CI_Type TempCI;
 Tsmall TempTEST;

 M93x6_Init();

 for( Adrs = 0; Adrs < 256; Adrs++ )
  M93x6_Write( Adrs, 0xFF );

 for( Adrs = 0; Adrs < 256; Adrs++ )
  if( M93x6_Read( Adrs ) != 0xFF )
   return;

 ConfigWrite( &(*ConfigRefCounter), (void *)&RefCounter, sizeof(
MB15E03_RefCounterType ) );
 for( Row = 0; Row < MAX_CHNLS; Row++ )
  for( Column = 0; Column < 2; Column++ )
```

```
    ConfigWrite( &(*ConfigCounter)[Row][Column], (void *)&Counter[Row][Column], sizeof(
MB15E03_CounterType ) );
 for( Row = 0; Row < MAX_CHNLS; Row++ )
  for( Column = 0; Column < MAX_PN_SEQS; Column++ )
   ConfigWrite( &(*ConfigPN_Codes)[Row][Column], (void *)&PN_Codes[Row][Column],
sizeof( Tlarge ) );
 for( Row = 0; Row < MAX_CHNLS; Row++ )
  ConfigWrite( &(*ConfigUW)[Row], (void *)&UW[Row], sizeof( Tlarge ) );
 ConfigWrite( &(*ConfigCI), (void *)&CI, sizeof( AIC9001_CI_Type ) );
 ConfigWrite( &(*ConfigTEST), (void *)&TEST, sizeof( Tsmall ) );


 ConfigRead( &TempRefCounter, &(*ConfigRefCounter), sizeof( MB15E03_RefCounterType )
);
 for( Row = 0; Row < MAX_CHNLS; Row++ )
  for( Column = 0; Column < 2; Column++ )
   ConfigRead( &TempCounter, &(*ConfigCounter)[Row][Column], sizeof(
MB15E03_CounterType ) );
 for( Row = 0; Row < MAX_CHNLS; Row++ )
  for( Column = 0; Column < MAX_PN_SEQS; Column++ )
   ConfigRead( &TempPN_Codes, &(*ConfigPN_Codes)[Row][Column], sizeof( Tlarge ) );
 for( Row = 0; Row < MAX_CHNLS; Row++ )
  ConfigRead( &TempUW, &(*ConfigUW)[Row], sizeof( Tlarge ) );
 ConfigRead( &TempCI, &(*ConfigCI), sizeof( AIC9001_CI_Type ) );
 ConfigRead( &TempTEST, &(*ConfigTEST), sizeof( Tsmall ) );
}
```

```
#ifndef CONFIG_H
#define CONFIG_H

/*****************************************************************************/
/*                                                                           */
/* Configuration Parameter Module                                            */
/* ------------------------------                                            */
/*                                                                           */
/* File: ..\config\config.h                                                  */
/* Date: 99.10.27                                                            */
/* By:   Arch                                                                */
/* Description:                                                              */
/*                                                                           */
/* The Configuration Paramter Module facilitates reading and                 */
/* writing data structures from and to a serial EEPROM device.               */
/*                                                                           */
/*****************************************************************************/

// INCLUDES
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\chnl\chnl.h>
#include <..\mb15e03\mb15e03.h>
#include <..\aic9001\aic9001.h>

// VARIABLES
extern MB15E03_RefCounterType (idata * const ConfigRefCounter);
extern MB15E03_CounterType (idata * const ConfigCounter)[MAX_CHNLS][2];
extern Tlarge (idata * const ConfigPN_Codes)[MAX_CHNLS][MAX_PN_SEQS];
extern Tlarge (idata * const ConfigUW)[MAX_CHNLS];
extern AIC9001_CI_Type (idata * const ConfigCI);
extern Tsmall (idata * const ConfigTEST);

// PROTOTYPES
void ConfigRead( void *Dst, void *Src, Tsmall Len );
void ConfigWrite( void *Dst, void *Src, Tsmall Len );

#endif
```

```
#ifndef __H82128__
#define __H82128__

/*********************************************************************/
/*                                                                   */
/* Ports and interrupts for the Hitachi H82128 microcontroller  */
/*                                                                   */
/*********************************************************************/

/* Types */
typedef volatile char ICDR_Type;
typedef volatile struct
{
 unsigned SVA:7;
 unsigned FS:1;
 const unsigned :8;
} SAR_Type;
typedef volatile struct
{
 unsigned MLS:1;
 unsigned WAIT:1;
 unsigned CKS:3;
 unsigned BC:3;
 const unsigned :8;
} ICMR_Type;
typedef volatile struct
{
 unsigned ICE:1;
 unsigned ICIE:1;
 unsigned MST:1;
 unsigned TRS:1;
 unsigned ACKE:1;
 unsigned BBSY:1;
 unsigned IRIC:1;
 unsigned SCP:1; /* Write only!!! */
 const unsigned :8;
} ICCR_Type;
typedef volatile struct
{
 unsigned ESTP:1;
 unsigned STOP:1;
 unsigned IRTR:1;
 unsigned AASX:1;
 unsigned AL:1;
 unsigned AAS:1;
```

```
  unsigned ADZ:1;
  unsigned ACKB:1;
  const unsigned :8;
 } ICSR_Type;
 typedef volatile struct
 {
  unsigned SWE:1;
  unsigned SW:1;
  unsigned IE:1;
  unsigned IF:1;
  unsigned CLR4:4;
  const unsigned :8;
 } DDCSWR_Type;

 typedef volatile const char RDR_Type;
 typedef volatile char TDR_Type;
 typedef volatile char BRR_Type;
 typedef volatile struct
 {
  unsigned CA:1;
  unsigned CHR:1;
  unsigned PE:1;
  unsigned OE:1;
  unsigned STOP:1;
  unsigned MP:1;
  unsigned CKS:2;
  const unsigned :8;
 } SMR_Type;
 typedef volatile struct
 {
  unsigned TIE:1;
  unsigned RIE:1;
  unsigned TE:1;
  unsigned RE:1;
  unsigned MPIE:1;
  unsigned TEIE:1;
  unsigned CKE:2;
  const unsigned :8;
 } SCR_Type;
typedef volatile struct
{
 unsigned TDRE:1;
 unsigned RDRF:1;
 unsigned ORER:1;
 unsigned FER:1;
```

```
    unsigned PER:1;
    const unsigned TEND:1;
    const unsigned MPB:1;
    unsigned MPBT:1;
    const unsigned :8;
  } SSR_Type;
  typedef volatile struct
  {
    const unsigned :4;
    unsigned SDIR:1;
    unsigned SINV:1;
    const unsigned :1;
    unsigned SMIF:1;
    const unsigned :8;
  } SCMR_Type;
  typedef volatile struct
  {
    const unsigned :1;
    unsigned IICX1:1;
    unsigned IICX0:1;
    unsigned IICE:1;
    unsigned FLSHE:1;
    const unsigned :1;
    unsigned ICKS:2;
  } STCR_Type;
  typedef volatile struct
  {
    unsigned B15:1;
    unsigned B14:1;
    unsigned P13:1;
    unsigned B12:1;
    unsigned B11:1;
    unsigned B10:1;
    unsigned B9:1;
    unsigned B8:1;
    unsigned B7:1;
    unsigned B6:1;
    unsigned B5:1;
    unsigned B4:1;
    unsigned B3:1;
    unsigned B2:1;
    unsigned B1:1;
    unsigned B0:1;
  } MSTPCR_Type;
  typedef volatile struct
```

```
  {
   unsigned B7:1;
   unsigned B6:1;
   unsigned B5:1;
   unsigned B4:1;
   unsigned B3:1;
   unsigned B2:1;
   unsigned B1:1;
   unsigned B0:1;
   const unsigned :8;
  } PortType;
  typedef volatile struct
  {
   const unsigned :5;
   unsigned B2:1;
   unsigned B1:1;
   unsigned B0:1;
   const unsigned :8;
  } Port5_Type;

  /* Interrupt vectors */
  #define ERI1 0x000150
  #define RXI1 0x000154
  #define TXI1 0x000158
  #define TEI1 0x00015C


  #ifdef __GNU97R1A__

  /* Register */
  #define DDCSWR   (*(DDCSWR_Type *)  (0xFFFEE6)) /* DDC switch register          */
  #define ICCR1    (*(ICCR_Type *)    (0xFFFF88)) /* I2C bus control register     */
  #define ICSR1    (*(ICSR_Type *)    (0xFFFF89)) /* I2C bus status register      */
  #define ICDR1    (*(ICDR_Type *)    (0xFFFF8E)) /* I2C bus data register        */
  #define SARX1    (*(SAR_Type *)     (0xFFFF8E)) /* I2C second slave address register */
  #define ICMR1    (*(ICMR_Type *)    (0xFFFF8F)) /* I2C bus mode register        */
  #define SAR1     (*(SAR_Type *)     (0xFFFF8F)) /* I2C slave address register   */

  #define ICCR0    (*(ICCR_Type *)    (0xFFFFD8)) /* I2C bus control register     */
  #define ICSR0    (*(ICSR_Type *)    (0xFFFFD9)) /* I2C bus status register      */
  #define ICDR0    (*(ICDR_Type *)    (0xFFFFDE)) /* I2C bus data register        */
  #define SARX0    (*(SAR_Type *)     (0xFFFFDE)) /* I2C second slave address register */
  #define ICMR0    (*(ICMR_Type *)    (0xFFFFDF)) /* I2C bus mode register        */
  #define SAR0     (*(SAR_Type *)     (0xFFFFDF)) /* I2C slave address register   */
```

```
#define MSTPCR   (*(MSTPCR_Type *)  (0xFFFF86)) /* Serial timer control register      */
#define STCR     (*(STCR_Type *)    (0xFFFFC3)) /* Serial timer control register      */

#define SMR0     (*(SMR_Type *)     (0xFFFFD8)) /* Serial mode register         ch 0 */
#define BRR0     (*(BRR_Type *)     (0xFFFFD9)) /* Bit rate register            ch 0 */
#define SCR0     (*(SCR_Type *)     (0xFFFFDA)) /* Serial control register      ch 0 */
#define TDR0     (*(TDR_Type *)     (0xFFFFDB)) /* Transmit data register       ch 0 */
#define SSR0     (*(SSR_Type *)     (0xFFFFDC)) /* Serial status register       ch 0 */
#define RDR0     (*(RDR_Type *)     (0xFFFFDD)) /* Receive data register        ch 0 */
#define SCMR0    (*(SCMR_Type *)    (0xFFFFDE)) /* Serial interface mode register ch 0 */

#define SMR1     (*(SMR_Type *)     (0xFFFF88)) /* Serial mode register         ch 1 */
#define BRR1     (*(BRR_Type *)     (0xFFFF89)) /* Bit rate register            ch 1 */
#define SCR1     (*(SCR_Type *)     (0xFFFF8A)) /* Serial control register      ch 1 */
#define TDR1     (*(TDR_Type *)     (0xFFFF8B)) /* Transmit data register       ch 1 */
#define SSR1     (*(SSR_Type *)     (0xFFFF8C)) /* Serial status register       ch 1 */
#define RDR1     (*(RDR_Type *)     (0xFFFF8D)) /* Receive data register        ch 1 */
#define SCMR1    (*(SCMR_Type *)    (0xFFFF8E)) /* Serial interface mode register ch 1 */

#define P1PCR    (*(PortType *)     (0xFFFFAC)) /* Pull-up control register     port 1 */
#define P2PCR    (*(PortType *)     (0xFFFFAD)) /* Pull-up control register     port 2 */
#define P3PCR    (*(PortType *)     (0xFFFFAE)) /* Pull-up control register     port 3 */
#define P1DDR    (*(PortType *)     (0xFFFFB0)) /* Data directon register       port 1 */
#define P2DDR    (*(PortType *)     (0xFFFFB1)) /* Data directon register       port 2 */
#define P1DR     (*(PortType *)     (0xFFFFB2)) /* Data register                port 1 */
#define P2DR     (*(PortType *)     (0xFFFFB3)) /* Data register                port 2 */
#define P3DDR    (*(PortType *)     (0xFFFFB4)) /* Data directon register       port 3 */
#define P4DDR    (*(PortType *)     (0xFFFFB5)) /* Data directon register       port 4 */
#define P3DR     (*(PortType *)     (0xFFFFB6)) /* Data register                port 3 */
#define P4DR     (*(PortType *)     (0xFFFFB7)) /* Data register                port 4 */
#define P5DDR    (*(Port5_Type *)   (0xFFFFB8)) /* Data directon register       port 5 */
#define P6DDR    (*(PortType *)     (0xFFFFB9)) /* Data directon register       port 6 */
#define P5DR     (*(Port5_Type *)   (0xFFFFBA)) /* Data register                port 5 */
#define P6DR     (*(PortType *)     (0xFFFFBB)) /* Data register                port 6 */

#else
extern DDCSWR_Type DDCSWR;
extern ICCR_Type ICCR1;
extern ICSR_Type ICSR1;
extern ICDR_Type ICDR1;
extern SAR_Type SARX1;
extern ICMR_Type ICMR1;
extern SAR_Type SAR1;

extern ICCR_Type ICCR0;
```

```
extern ICSR_Type ICSR0;
extern ICDR_Type ICDR0;
extern SAR_Type SARX0;
extern ICMR_Type ICMR0;
extern SAR_Type SAR0;

extern MSTPCR_Type MSTPCR;
extern STCR_Type STCR;

extern SMR_Type SMR0;          /* Serial mode register        ch 0 */
extern BRR_Type BRR0;          /* Bit rate register           ch 0 */
extern SCR_Type SCR0;          /* Serial control register     ch 0 */
extern TDR_Type TDR0;          /* Transmit data register      ch 0 */
extern SSR_Type SSR0;          /* Serial status register      ch 0 */
extern RDR_Type RDR0;          /* Receive data register       ch 0 */
extern SCMR_Type SCMR0;        /* Serial interface mode register ch 0 */

extern SMR_Type SMR1;          /* Serial mode register        ch 1 */
extern BRR_Type BRR1;          /* Bit rate register           ch 1 */
extern SCR_Type SCR1;          /* Serial control register     ch 1 */
extern TDR_Type TDR1;          /* Transmit data register      ch 1 */
extern SSR_Type SSR1;          /* Serial status register      ch 1 */
extern RDR_Type RDR1;          /* Receive data register       ch 1 */
extern SCMR_Type SCMR1;        /* Serial interface mode register ch 1 */

extern PortType P1PCR;         /* Pull-up control register    port 1 */
extern PortType P2PCR;         /* Pull-up control register    port 2 */
extern PortType P3PCR;         /* Pull-up control register    port 3 */
extern PortType P1DDR;         /* Data directon register      port 1 */
extern PortType P2DDR;         /* Data directon register      port 2 */
extern PortType P1DR;          /* Data register               port 1 */
extern PortType P2DR;          /* Data register               port 2 */
extern PortType P3DDR;         /* Data directon register      port 3 */
extern PortType P4DDR;         /* Data directon register      port 4 */
extern PortType P3DR;          /* Data register               port 3 */
extern PortType P4DR;          /* Data register               port 4 */
extern PortType P5DDR;         /* Data directon register      port 5 */
extern PortType P6DDR;         /* Data directon register      port 6 */
extern PortType P5DR;          /* Data register               port 5 */
extern PortType P6DR;          /* Data register               port 6 */

#endif

#endif
```

```
#ifndef I8051SCI_H
#define I8051SCI_H

#include <..\std_inc\cc_std.h>

void i8051SciInit( void );
Tbool i8051SciRead( Tsmall *RxByte );
Tbool i8051SciWrite( Tsmall TxByte );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\mcu_def\i8xc32.h>
#include <..\queue\queue.h>
#include <..\i8051sci\i8051sci.h>

#if ( __I8051SCI_BAUD_GEN_TIMER__ == 1 ) // timer 1 mode 2.
 #if ( ( 2 * I8051SCI_BAUD ) > XTAL_FREQ / 12.0 / 32.0 )
  #define RELOAD    ( 256 - ( 2 * XTAL_FREQ / 12.0 / 32.0 / I8051SCI_BAUD ) )
  #define SMOD
 #else
  #define RELOAD    ( 256 - ( XTAL_FREQ / 12.0 / 32.0 / I8051SCI_BAUD ) )
 #endif
#endif
#if ( __I8051SCI_BAUD_GEN_TIMER__ == 2 )
 #define RELOAD      ( 65536L - ( XTAL_FREQ / 32.0 / I8051SCI_BAUD ) )
 #define RELOAD_HI   ( RELOAD / 256 )
 #define RELOAD_LO   ( RELOAD )
#endif

Tbool i8051SciTxBusy;

/*****************************************************************************/
/*                                                                           */
/* QUEUE_CREATE( Tx, I8051SCI_TX_Q_SIZE )                                    */
/*---------------------------------                                          */
/*                                                                           */
/* This invocation allocates memory and declares functions for               */
/* a queue of characters I8051SCI_TX_Q_SIZE bytes long named Tx.             */
/* Functions TxWrite and TxRead are automatically declared that              */
/* work directly on this queue. The prototypes are as follows:               */
/*                                                                           */
/* Tbool TxWrite( Tsmall Data );                                             */
/* Tbool TxRead( Tsmall *Data );                                             */
/*                                                                           */
/* The write function writes Data to the queue, while the read               */
/* function removes the oldest element from the queue and writes             */
/* it to the location specified by Data. Both functions return               */
/* booleans to indicate the success of the respective operation.             */
/*                                                                           */
/*****************************************************************************/
QUEUE_CREATE( Tx, I8051SCI_TX_Q_SIZE )
QUEUE_CREATE( Rx, I8051SCI_RX_Q_SIZE ) // similar to above comment.

/*****************************************************************************/
```

```
/*                                                                    */
/* i8051SciInit()                                                     */
/* --------------                                                     */
/*                                                                    */
/* This function initializes the module and associated hardware.      */
/* It uses either TIMER 1 or TIMER 2 depending on the value of the    */
/* __I8051SCI_BAUD_GEN__ definition. The I8051SCI_DOUBLE_BAUD         */
/* definition will set the SMOD bit in the PCON register.             */
/*                                                                    */
/**********************************************************************/
void i8051SciInit( void )
{
 i8051SciTxBusy = FALSE;

 QueueInit( Tx );
 QueueInit( Rx );
 SCON_SM1 = 1;          /* serial mode 2 (1, 8, 1).      */
 SCON_SM2 = 1;          /* valid stop bit required.      */
 SCON_REN = 1;          /* receiver enabled.             */
#ifdef SMOD
 PCON |= 0x80;
#endif
#if ( __I8051SCI_BAUD_GEN_TIMER__ == 1 )
 TMOD = ( TMOD & 0x0F ) | 0x20; /* timer 1 mode 2.          */
 TL1 = RELOAD;
 TH1 = RELOAD;
 TCON_TR1 = SET;
#endif
#if ( __I8051SCI_BAUD_GEN_TIMER__ == 2 )
 RCAP2H = RELOAD_HI;    /* set reload value.             */
 RCAP2L = RELOAD_LO;    /*                               */
 T2CON_RCLK = SET;      /* timer 2 baud rate generator.  */
 T2CON_TCLK = SET;
 T2CON_TR2 = SET;       /* start timer.                  */
#endif
 IE_ES = SET;           /* Rx/Tx serial interrupt enabled. */
}

/**********************************************************************/
/*                                                                    */
/* i8051SciRead()                                                     */
/* --------------                                                     */
/*                                                                    */
/* This function reads the next byte from the Rx queue (if            */
/* available), and returns it to the location specified by RxByte.    */
```

```
 /* It returns a boolean to indicate whether a byte was indeed      */
 /* available.                                                       */
 /*                                                                  */
 /********************************************************************/
 Tbool i8051SciRead( Tsmall *RxByte )
 {
  return( RxRead( RxByte ) );
 }

 /********************************************************************/
 /*                                                                  */
 /* i8051SciWrite()                                                  */
 /* ---------------                                                  */
 /*                                                                  */
 /* This function writes TxByte to the Tx queue (if not already      */
 /* full). It returns a boolean to indicate the success of the       */
 /* operation.                                                       */
 /*                                                                  */
 /********************************************************************/
 Tbool i8051SciWrite( Tsmall TxByte )
 {
  if( !TxWrite( TxByte ) ) /* if no TxByte available...             */
    return( FALSE );       /*   return FALSE.                       */
  if( !i8051SciTxBusy )
  {
    DisableInt();
    SCON_TI = SET;         /* Start transmitter, force vector */
    i8051SciTxBusy = TRUE;
    EnableInt();
  }
  return( TRUE );          /* return TRUE.                        */
 }

 /********************************************************************/
 /*                                                                  */
 /* i8051SciISR()                                                    */
 /* -------------                                                    */
 /*                                                                  */
 /* This ISR services the receiver and transmitter flags.            */
 /*                                                                  */
 /********************************************************************/
 interrupt [0x23] void i8051SciISR( void )
 {
  Tsmall TxByte;
```

```
 if( SCON_RI )          /* if receiver flag...          */
 {
  SCON_RI = CLEAR;      /*  kill receiver flag.         */
  RxWrite( SBUF );      /*  write SBUF to Rx queue.     */
 }
 if( SCON_TI )          /* if transmitter flag...       */
 {
  SCON_TI = CLEAR;      /*  Kill transmitter flag.      */
  if( TxRead( &TxByte ) ) /*  if TxByte available...    */
   SBUF = TxByte;       /*    write TxByte to SBUF.     */
  else i8051SciTxBusy = FALSE;
 }
}
```

```
#ifndef HC594_H
#define HC594_H

#include <..\std_inc\cc_std.h>

void HC594_Write( Tsmall Data );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\mcu_def\i8xc32.h>
#include <..\HC594\HC594.h>

// FUNCTIONS
/*************************************************************************/
/*                                                                       */
/* HC594_Write()                                                         */
/* -------------                                                         */
/*                                                                       */
/* This function performs serial data transfer to a 74hc594              */
/* buffered shift register.  It writes a byte to the device MSB          */
/* first.  It performs data clocking, device selection, etc.             */
/*                                                                       */
/*************************************************************************/
void HC594_Write( Tsmall Data )
{
 Tsmall NumBits = 8;

 HC594_CS_N = ASSERT_N;
 while( NumBits-- )
 {
  HC594_CLK = CLEAR;
  if( Data & 0x80 )
    HC594_DATA = SET;
  else HC594_DATA = CLEAR;
  HC594_CLK = SET;
  Data <<= 1;
 }
 HC594_CLK = CLEAR;
 HC594_DATA = SET;
 HC594_CS_N = !ASSERT_N;
}
```

```
#ifndef HC55181_H
#define HC55181_H

void HC55181_Init( void );
void HC55181_Run( void );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\mcu_def\i8xc32.h>
#include <..\status\status.h>
#include <..\hc594\hc594.h>
#include <..\hc55181\hc55181.h>

// DEFINITIONS
#define RELOAD   ( -(Tmedium)( XTAL_FREQ / 12 * HC55181_UPDATE_TIME - 96 ) )
#define DEBOUNCE 60 // ~15 ms

// TYPES
typedef enum
{
 ASSERTED,
 DEASSERTED = ASSERTED + DEBOUNCE
} HC55181_FilteredDetectType;

typedef enum
{
 VBL,
 VBH
} HC55181_BatteryType;

typedef enum
{
 NOT_RINGING,
 RING,
 RING_TRIP,
 LATCH_CLEAR,
 OFF_HOOK
} HC55181_RingStateType;

typedef enum
{
 LOW_POWER_STANDBY,
 FORWARD_ACTIVE = LOW_POWER_STANDBY + HC55181_MODE_STEP,
 UNUSED = FORWARD_ACTIVE + HC55181_MODE_STEP,
 REVERSE_ACTIVE = UNUSED + HC55181_MODE_STEP,
 RINGING = REVERSE_ACTIVE + HC55181_MODE_STEP,
 FORWARD_LOOP_BACK = RINGING + HC55181_MODE_STEP,
 TIP_OPEN = FORWARD_LOOP_BACK + HC55181_MODE_STEP,
 POWER_DENIAL = TIP_OPEN + HC55181_MODE_STEP
} HC55181_OperatingModeType;
```

```
// VARIABLES
Tsmall HC55181_Mode = FORWARD_ACTIVE | VBH;

// FUNCTIONS
void HC55181_Init( void )
{
 TMOD = ( TMOD & 0xF0 ) | 0x01;       // timer 0 mode 1.
 TCON_TF0 = 1;                        // cause vector.
 IE_ET0 = 1;                          // enable interrupts.
 TCON_TR0 = SET;                      // enable timer run.
}

void HC55181_Run( void )
{
 static HC55181_FilteredDetectType FilteredDetect = DEASSERTED;
 static HC55181_RingStateType RingState;
 Tsmall Mode;
 Tsmall RxStatus;

 if( HC55181_DET_N == ASSERT_N )      // if detect is asserted...
  FilteredDetect = ASSERTED;          //  filtered detect follows instantly
 else                                 // else
  if( FilteredDetect != DEASSERTED )  //  increment filtered detect up to DEASSERTED
   FilteredDetect++;
 StatusReadRx( &RxStatus );           // Get received status
 if( !( RxStatus & STATUS_RING_N ) )  // if line-device requests ringing...
 {
  switch( RingState )                 // case ring state of:
  {
   case NOT_RINGING:                  //  NOT RINGING
    RingState = RING;                 //   initialize ring state to RING
   case RING:                         //  RING
    HC55181_Mode = RINGING | VBH;     //   select RINGING mode and VBH
    StatusWriteTx( 0xFF );            //   send "on-hook" status
    if( HC55181_DET_N == ASSERT_N )   //   if ring-trip occurred...
     RingState = RING_TRIP;           //    ring state <- RING-TRIP
    break;
   case RING_TRIP:                    //  RING-TRIP
    HC55181_Mode = FORWARD_ACTIVE | VBL;// cause latch-clear glitch
    if( HC55181_DET_N == !ASSERT_N )  //   if latch-clear glitch observed...
     RingState = LATCH_CLEAR;         //    ring state <- LATCH-CLEARED
    break;
   case LATCH_CLEAR:                  //  LATCH-CLEARED
    if( HC55181_DET_N == ASSERT_N )   //   if off-hook observed...
     RingState = OFF_HOOK;            //    ring-state <- OFF-HOOK
```

```
    break;
   case OFF_HOOK:                  //  OFF-HOOK
    StatusWriteTx( ~STATUS_OFF_HOOK_N );// send "off-hook" status
    break;
   default:;
  }
 }
 else                            // else
 {
  RingState = NOT_RINGING;
  if( FilteredDetect == DEASSERTED )  //  if on-hook...
  {
   Mode = VBH;                   //   select VBH
   StatusWriteTx( 0xFF );          //    send "on-hook" status
  }
  else                           //  else
  {
   Mode = VBL;                   //   select VBL
   StatusWriteTx( ~STATUS_OFF_HOOK_N );//   send "off-hook" status
  }
  if( RxStatus & STATUS_TR_REVERSED_N )//  if line-device requests TR normal...
   HC55181_Mode = FORWARD_ACTIVE | Mode;//  set slic mode to FORWARD_ACTIVE
  else HC55181_Mode = REVERSE_ACTIVE | Mode;//else set slic mode to
REVERSE_ACTIVE
 }
}

interrupt [0x0B] void HC55181_ISR( void )
{
 static Tsmall SineCntr = 0;
 Tsmall Data;

 TH0 = RELOAD / 256;               // reload timer.
 TL0 = RELOAD % 256;
 SineCntr += HC55181_SINE_STEP;      // increment sine counter.
 Data = 0x00;                    // if sine counter MSB set
 if( SineCntr & (HC55181_MODE_STEP>>1) )//  Data <- 0x00
  Data--;                        // else Data <- 0xFF
 Data ^= SineCntr;                 // Data <- Data XOR sine counter
 Data &= ( HC55181_MODE_STEP - HC55181_SINE_STEP );// Mask-off other bits
 Data |= HC55181_Mode;             // add mode bits and set battery
 HC594_Write( Data );              // output all bits
}
```

```
#include <..\std_inc\cc_std.h>
#include <..\aic9001\aic9001.h>
#include <..\chnl\chnl.h>
#include <..\config\config.h>
#include <..\mb15e03\mb15e03.h>

// DEFINITIONS
/*****************************************************************************/
/*                                                                           */
/* CHNL_RATIOS( Chnl )                                                       */
/* -------------------                                                       */
/*                                                                           */
/* This macro provides PLL divider ratio data for both the receive           */
/* and transmit frequencies of the specified radio channel (Chnl)            */
/* based on the parameters defined in the mb15e03.h file.                    */
/*                                                                           */
/*****************************************************************************/

// VARIABLES
MB15E03_CounterType Cntr[2];     // Cntr buffers.

// FUNCTIONS
/*****************************************************************************/
/*                                                                           */
/* MB15E03_SerialWrite()                                                     */
/* ---------------------                                                     */
/*                                                                           */
/* This function performs serial data transfer to the device's               */
/* synchronous serial port. It sends the NumBits of the MSB's of             */
/* Data. Data bits are transferred MSB first. It performs only               */
/* data clocking. Device selection and data latching is performed            */
/* by an autonomous hardware signal. When the signal is low,                 */
/* the device is ready to accept data. The signal must be fed to             */
/* INT0_N to allert the CPU of the opportunity to write data.                */
/* The data is latched into the device when the hardware signal              */
/* returns high.                                                             */
/*                                                                           */
/*****************************************************************************/
void MB15E03_SerialWrite( Tmedium Data, Tsmall NumBits )
{
 while( NumBits-- )
 {
  MB15E03_CLOCK = CLEAR;
  if( Data & 0x8000 )
   MB15E03_DATA_OUT = SET;
```

```
    else MB15E03_DATA_OUT = CLEAR;
    MB15E03_CLOCK = SET;
    Data <<= 1;
  }
  MB15E03_CLOCK = CLEAR;
  MB15E03_DATA_OUT = SET;
}

/*************************************************************************/
/*                                                                       */
/* MB15E03_Init()                                                        */
/* --------------                                                        */
/*                                                                       */
/* This function initializes the MB15E03 device I/O.                     */
/*                                                                       */
/*************************************************************************/
void MB15E03_Init( void )
{
  MB15E03_INT_N = SET;          // input.
  MB15E03_DATA_OUT = SET;       // idle state.
  MB15E03_CLOCK = CLEAR;        // idle state.
}

/*************************************************************************/
/*                                                                       */
/* MB15E03_Mode()                                                        */
/* --------------                                                        */
/*                                                                       */
/* This function initializes the MB15E03 mode of operation. It           */
/* buffers the receive and transmit programmable counter data in         */
/* preparation for rapid frequency programming. It programs the          */
/* reference counter register, and programs the counter register         */
/* for the first frequency used Tx/Rx for Master/Slave respectively.     */
/* Finally, it enables INT0_N for falling edge sensitivity at high       */
/* priority.                                                             */
/*                                                                       */
/*************************************************************************/
void MB15E03_Mode( M_SN_Type Data )
{
  MB15E03_RefCounterType RefCounter;

  IE_EX0 = 0;                   // INT0_N disabled.
  POWER_STANDBY_N = !ASSERT_N;  // enable 4Vreg.
  ConfigRead( &Cntr, &(*ConfigCounter)[Chnl][0], sizeof( MB15E03_CounterType ) * 2 );
  Cntr[0].N <<= 5;
```

```
Cntr[0].A <<= 1;
Cntr[1].N <<= 5;
Cntr[1].A <<= 1;
MB15E03_SELECT;
ConfigRead( &RefCounter, &(*ConfigRefCounter), sizeof( MB15E03_RefCounterType ) );
MB15E03_SerialWrite( *(Tmedium *)&RefCounter.CntrlBits, 4 );
MB15E03_SerialWrite( *(Tmedium *)&RefCounter.R<<2, 14 );
MB15E03_SerialWrite( REF_COUNTER, 1 );
MB15E03_DESELECT;
MB15E03_SELECT;
if( Data == MASTER )          // if device is MASTER
{                             //  transmit first
 MB15E03_SerialWrite( *(Tmedium *)&Cntr[1].N, 11 );
 MB15E03_SerialWrite( *(Tmedium *)&Cntr[1].A, 8 );
}
else                          // else receive first.
{
 MB15E03_SerialWrite( *(Tmedium *)&Cntr[0].N, 11 );
 MB15E03_SerialWrite( *(Tmedium *)&Cntr[0].A, 8 );
}
MB15E03_DESELECT;
TCON_IT0 = 1;                 // INT0_N falling edge sensitive.
IP_PX0 = 1;                   // INT0_N high priority.
TCON_IE0 = 0;                 // INT0_N clear.
IE_EX0 = 1;                   // INT0_N enabled.
}

/*************************************************************************/
/*                                                                       */
/* MB15E03_ISR()                                                         */
/* -------------                                                         */
/*                                                                       */
/* This ISR services the INT0_N flag.  When set, the device is           */
/* ready to accept serial programming.  The AIC9001_PLLSW line           */
/* indicates whether the device is currently transmitting or             */
/* receiving.                                                            */
/*                                                                       */
/*************************************************************************/
interrupt [0x03] void MB15E03_ISR( void )
{
 if( AIC9001_PLLSW == ASSERT )   // if transmitting
 {                               //  clock-in next rcv freq.
  MB15E03_SerialWrite( *(Tmedium *)&Cntr[0].N, 11 );
  MB15E03_SerialWrite( *(Tmedium *)&Cntr[0].A, 8 );
 }
```

```
  else                    // else next xmit freq.
  {
   MB15E03_SerialWrite( *(Tmedium *)&Cntr[1].N, 11 );
   MB15E03_SerialWrite( *(Tmedium *)&Cntr[1].A, 8 );
  }
}
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\mcu_def\i8xc32.h>
#include <..\m93x6\m93x6.h>
#include <..\i8051sci\i8051sci.h>

void main( void )
{
 Tsmall Count;

 M93x6_Init();
 i8051SciInit();
 M93x6_Write( 0, 0x98 );
 M93x6_Write( 2, 0x76 );
 M93x6_Write( 3, 0x54 );
 M93x6_Write( 4, 0x32 );
 M93x6_Write( 1, 0x10 );
 for( Count = 0; Count < 5; Count++ )
  i8051SciWrite( M93x6_Read( Count ) );
 for( Count = 0; Count < 5; Count++ )
  M93x6_Write( Count, 0xFF );
 for( Count = 0; Count < 5; Count++ )
  i8051SciWrite( M93x6_Read( Count ) );
}
```

```
#ifndef M93X6_H
#define M93X6_H

#include <..\std_inc\cc_std.h>
#include <..\mcu_def\i8xc32.h>

#define M93X6_MAX_COUNT 1000

void M93x6_Init( void );
Tsmall M93x6_Read( Tmedium Adrs );
Tbool M93x6_Write( Tmedium Adrs, Tsmall Data );

#endif
```

```
/*************************************************************/
/*                                                           */
/* Byte Addressable M93x6 Module                             */
/* ----------------------------                              */
/*                                                           */
/* File: ..\m93x6\m93x6.c                                    */
/* Date: 99.10.20                                            */
/* By:   Arch                                                */
/* Description:                                              */
/*                                                           */
/* This module allows byte addressable random access         */
/* (reads and writes) of a Microchip 93xx serial             */
/* EEPROM device.                                            */
/*                                                           */
/*************************************************************/

// INCLUDES
#include <..\std_inc\compiler.h>
#include <..\m93x6\m93x6.h>

// DEFINITIONS
#define M93X6_SB          0x04
#define M93X6_OPCODE_SIZE   3
#define WRITE_DISABLE     (EWDS<<( M93X6_NUM_ADRS_BITS - 2 ))
#define WRITE_ALL         (WRAL<<( M93X6_NUM_ADRS_BITS - 2 ))
#define ERASE_ALL         (ERAL<<( M93X6_NUM_ADRS_BITS - 2 ))
#define WRITE_ENABLE      (EWEN<<( M93X6_NUM_ADRS_BITS - 2 ))

// TYPES
typedef enum
{
 ALL_ADRS = M93X6_SB,
 WRITE,
 READ,
 ERASE
} M93x6_OpcodeType;

typedef enum
{
 EWDS,
 WRAL,
 ERAL,
 EWEN
} M93x6_SpecialAdrsType;
```

```
// FUNCTIONS
/*************************************************************/
/*                                                           */
/* M93x6_Init()                                              */
/* ------------                                              */
/*                                                           */
/* This function initializes the module and device.          */
/*                                                           */
/*************************************************************/
void M93x6_Init( void )
{
 M93X6_SELECT = !ASSERT;
}

/*************************************************************/
/*                                                           */
/* M93x6_Receive()                                           */
/* ---------------                                           */
/*                                                           */
/* This function performs a synchronous serial transfer      */
/* from the device's serial port to the MCU. It performs     */
/* various hardware actions including device selection       */
/* and data clocking. It reads and returns eight             */
/* bits MSB first. It does NOT de-select the device.         */
/*                                                           */
/*************************************************************/
Tsmall M93x6_Receive( void )
{
 Tsmall NumBits = 8;
 Tsmall Data = 0;

 M93X6_DI = SET;
 M93X6_CLK = CLEAR;
 M93X6_SELECT = ASSERT;
 while( NumBits-- )
 {
  M93X6_CLK = SET;
  Data <<= 1;
  M93X6_CLK = CLEAR;
  if( M93X6_DO == SET )
    Data |= 0x01;
 }
 M93X6_DI = SET;
 return( Data );
}
```

```
/*************************************************************/
/*                                                           */
/* M93x6_Transmit()                                          */
/* ----------------                                          */
/*                                                           */
/* This function performs a synchronous serial transfer      */
/* from the MCU to the device's serial port. It performs     */
/* various hardware actions including device selection       */
/* and data clocking. It transmits NumBits of the MSB's      */
/* of Data. It does NOT de-select the device.                */
/*                                                           */
/*************************************************************/
void M93x6_Transmit( Tmedium Data, Tsmall NumBits )
{
 M93X6_CLK = CLEAR;
 M93X6_SELECT = ASSERT;
 Data <<= 16 - NumBits;
 while( NumBits-- )
 {
  if( Data & 0x8000 )
    M93X6_DI = SET;
  else M93X6_DI = CLEAR;
  M93X6_CLK = SET;
  Data <<= 1;
  M93X6_CLK = CLEAR;
 }
 M93X6_DI = SET;
}

/*************************************************************/
/*                                                           */
/* M93x6_Read()                                              */
/* ------------                                              */
/*                                                           */
/* This function reads and returns a byte from the           */
/* EEPROM location specified by Adrs.                        */
/*                                                           */
/*************************************************************/
Tsmall M93x6_Read( Tmedium Adrs )
{
 Tsmall Result;

 M93x6_Transmit( READ, M93X6_OPCODE_SIZE );
 M93x6_Transmit( Adrs, M93X6_NUM_ADRS_BITS );
 Result = M93x6_Receive();
```

```
 M93X6_SELECT = !ASSERT;
 return( Result );
}

/*************************************************************/
/*                                                           */
/* M93x6_Write()                                             */
/* -------------                                             */
/*                                                           */
/* This function writes Data (byte) to the EEPROM            */
/* location specified by Adsrs. The function returns a       */
/* boolean to indicate the function's respective success.    */
/*                                                           */
/*************************************************************/
Tbool M93x6_Write( Tmedium Adrs, Tsmall Data )
{
 Tmedium Count = M93X6_MAX_COUNT;

 M93x6_Transmit( ALL_ADRS, M93X6_OPCODE_SIZE );
 M93x6_Transmit( WRITE_ENABLE, M93X6_NUM_ADRS_BITS );
 M93X6_SELECT = !ASSERT;

 M93x6_Transmit( WRITE, M93X6_OPCODE_SIZE );
 M93x6_Transmit( Adrs, M93X6_NUM_ADRS_BITS );
 M93x6_Transmit( Data, 8 );
 M93X6_SELECT = !ASSERT;

 M93X6_SELECT = ASSERT;
 while( ( M93X6_DO == CLEAR ) && Count-- );
 M93X6_SELECT = !ASSERT;

 M93x6_Transmit( ALL_ADRS, M93X6_OPCODE_SIZE );
 M93x6_Transmit( WRITE_DISABLE, M93X6_NUM_ADRS_BITS );
 M93X6_SELECT = !ASSERT;

 return( Count > 0 );
}
```

```
//            -LNK8051.XCL-
//
//      XLINK 4.44, or higher, command file to be used with the 8051
//      C-compiler V5.xx
//      using the -mt, -ms, -mc, -mm or -ml memory model
//      Usage:  xlink  your_file(s)  -f lnk8051l
//
//      First: define CPU
//
//  Revision control system
//     $Id: lnk8051.xcl,v 1.1 1999-12-08 09:18:54-06 mlockerb Exp $
//


-c8051

// If you have register independent code use: -D_R=0
// (or 8,16,24) to choose the register bank used at startup
-D_R=0

// Setup "bit" segments (always zero if there is no need to reserve
//   bit variable space for some other purpose)

-Z(BIT)C_ARGB,BITVARS=0

// Setup "data" segments.  Start address may not be less
// than start of register bank + 8.  Space must also
// be left for interrupt functions with the "using" attribute.

-Z(DATA)B_UDATA,B_IDATA,C_ARGD,D_UDATA,D_IDATA=8

// Setup "idata" segments (usually loaded after "data")

-Z(IDATA)C_ARGI,I_UDATA,I_IDATA,CSTACK+20

// Setup "xdata" segments to the start address of external RAM.
// Note that it starts from 1 since a pointer to address zero is regarded
// as NULL.
// Note that this declaration does no harm even if you use a memory
// model that does not utilize external data RAM

-Z(XDATA)P_UDATA,P_IDATA,C_ARGX,X_UDATA,X_IDATA,ECSTR,RF_XDATA,XST
ACK=1

// Setup all read-only segments (PROM). Usually at zero
```

-Z(CODE)INTVEC,RCODE,D_CDATA,B_CDATA,I_CDATA,P_CDATA,X_CDATA,C_ICALL,C_RECFN,CSTR,CCSTR,CODE,CONST=0

// See configuration section concerning printf/sprintf
-e_small_write=_formatted_write

// See configuration section concerning scanf/sscanf
-e_medium_read=_formatted_read

// Load the 'C' library adapted for the selected memory model
// cl8051t , cl8051s, cl8051c, cl8051m, cl8051l

// Code will now reside on file aout.a03 in INTEL-STANDARD format

```
/*****************************************************************************/
/*                                                                           */
/* Millenium TelePath Line-side Device f/ware                                */
/* ------------------------------------                                      */
/*                                                                           */
/* This firmware implements the line-side functionality of the               */
/* Millenium TelePath wireless POTS device.                                  */
/*                                                                           */
/*****************************************************************************/
#include <..\std_inc\cc_std.h>     // standard CC types
#include <..\std_inc\compiler.h>   // compiler specific defines
#include <..\chnl\chnl.h>          // channel selection module
#include <..\ds1706\ds1706.h>      // CPU-supervisor module
#include <..\led\led.h>            // LED module

#define CLOCK P1_6
#define DATA  P1_5

void SerialWrite( Tsmall Data, Tsmall NumBits )
{
 while( NumBits-- )
 {
  CLOCK = CLEAR;
  if( Data & 0x80 )
   DATA = SET;
  else DATA = CLEAR;
  CLOCK = SET;
  Data <<= 1;
 }
 DATA = SET;
}

void main( void )
{
 Tmedium Delay;

 /*
  P0 = 0xAE; // make I/O safe.
  P1 = 0xEF;
  P2 = 0xFB;
  P3 = 0xFF;
 */
  P0 = 0xFF; // make I/O inputs (safe).
  P1 = 0xFF;
  P2 = 0xFF;
```

```
 P3 = 0xFF;

 while( 1 )
 {
  ChnlInit();
  SerialWrite( Chnl, 8 );
  for( Delay = 0; Delay < 1000; Delay++ );
  if( LED_N == ASSERT_N )
   LED_N = !ASSERT_N;
  else LED_N = ASSERT_N;
  DS1706_Reset();
 }
}
```

```
#ifndef LINE_INTF_H
#define LINE_INTF_H

/*************************************************************************/
/*                                                                       */
/* Telephony Line Interface Module                                       */
/* -------------------------------                                       */
/*                                                                       */
/* File: ..\lineintf\lineintf.h                                          */
/* Date: 99.09.20                                                        */
/* By:   Arch                                                            */
/* Description:                                                          */
/*                                                                       */
/* The Telephony Line Interface Module controls the telephone            */
/* line connected to the line-side device.  It monitors the              */
/* states of the ring-detect and tip/ring reversal-detect                */
/* circuits and sends status nibbles that reflect these states           */
/* to the remote device.  It also interprets status nibbles              */
/* received from the remote device and updates its control lines         */
/* (on/off hook) to reflect these states.                                */
/*                                                                       */
/* The LineIntfInit function initializes the module and its              */
/* respective control lines.                                             */
/*                                                                       */
/* The LineIntfRun function performs all module functionality            */
/* and should be executed rapidly in the main loop.                      */
/*                                                                       */
/*************************************************************************/

#include <..\mcu_def\i8xc32.h>

void LineIntfInit( void );
void LineIntfRun( void );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\status\status.h>
#include <..\lineintf\lineintf.h>

/*************************************************************************/
/*                                                                       */
/* LineIntfInit()                                                        */
/* --------------                                                        */
/*                                                                       */
/* This function initializes all line interface I/O.                     */
/*                                                                       */
/*************************************************************************/
void LineIntfInit( void )
{
 LINE_INTF_RINGDET_N = 1;        // input.
 LINE_INTF_T_R_REVERSED_N = 1;   // input.
 LINE_INTF_OFF_HOOK_N = !ASSERT_N;// line on-hook.
}

/*************************************************************************/
/*                                                                       */
/* LineIntfGet()                                                         */
/* -------------                                                         */
/*                                                                       */
/* This function monitors and debounces all associated line              */
/* interface inputs (including LINE_INTF_T_R_REVERSE_N and               */
/* LINE_INTF_RINGDET_N) and formulates a status nibble ready             */
/* for transmission to the phone-side device.                            */
/*                                                                       */
/*************************************************************************/
Tsmall LineIntfGet( void )
{
 static Tsmall RingCount = LINE_INTF_COUNT;
 static Tsmall FilteredRingDetectN = !ASSERT_N;
 static Tsmall TR_ReversedCount = LINE_INTF_COUNT;
 static Tsmall FilteredTR_ReversedN = !ASSERT_N;
 Tsmall Result;

 if( ( LINE_INTF_RINGDET_N == !ASSERT_N ) == FilteredRingDetectN )
 {
  if( FilteredRingDetectN == !ASSERT_N )
    RingCount = LINE_INTF_COUNT;
  else RingCount = LINE_INTF_RING_RELEASE_COUNT;
 }
```

```
 else
 {
  if( RingCount )
   RingCount--;
  else FilteredRingDetectN = !FilteredRingDetectN;
 }

 if( ( LINE_INTF_T_R_REVERSED_N == !ASSERT_N ) == FilteredTR_ReversedN )
  TR_ReversedCount = LINE_INTF_COUNT;
 else
 {
  if( TR_ReversedCount )
   TR_ReversedCount--;
  else FilteredTR_ReversedN = !FilteredTR_ReversedN;
 }
 Result = 0xFF;
 if( FilteredRingDetectN == ASSERT_N )
  Result &= ~STATUS_RING_N;
 if( FilteredTR_ReversedN == ASSERT_N )
  Result &= ~STATUS_TR_REVERSED_N;
 return( Result );
}
/*
Tsmall LineIntfGet( void )
{
 static Tsmall RingCount = LINE_INTF_COUNT;
 static bit FilteredRingDetectN = !ASSERT_N;
 static Tsmall TR_ReversedCount = LINE_INTF_COUNT;
 static bit FilteredTR_ReversedN = !ASSERT_N;
 Tsmall Result;

 if( LINE_INTF_RINGDET_N == FilteredRingDetectN )
 {
  if( FilteredRingDetectN == !ASSERT_N )
   RingCount = LINE_INTF_COUNT;
  else RingCount = LINE_INTF_RING_RELEASE_COUNT;
 }
 else
 {
  if( RingCount )
   RingCount--;
  else FilteredRingDetectN = !FilteredRingDetectN;
 }

 if( LINE_INTF_T_R_REVERSED_N == FilteredTR_ReversedN )
```

```
    TR_ReversedCount = LINE_INTF_COUNT;
   else
   {
    if( TR_ReversedCount )
     TR_ReversedCount--;
    else FilteredTR_ReversedN = !FilteredTR_ReversedN;
   }
   Result = 0xFF;
   if( FilteredRingDetectN == ASSERT_N )
    Result &= ~STATUS_RING_N;
   if( FilteredTR_ReversedN == ASSERT_N )
    Result &= ~STATUS_TR_REVERSED_N;
   return( Result );
  }
  */

  /*************************************************************************/
  /*                                                                       */
  /* LineIntfRun()                                                         */
  /* -------------                                                         */
  /*                                                                       */
  /* This function writes the new line interface status to the             */
  /* transmit status process in order to relay this information            */
  /* to the phone-side device.  It also reads the received status          */
  /* from the phone-side device and updates the line interface             */
  /* outputs (LINE_INTF_OFF_HOOK_N) to reflect this.                       */
  /*                                                                       */
  /*************************************************************************/
  void LineIntfRun( void )
  {
   Tsmall Status;

   // get outgoing status and write it.
   Status = LineIntfGet();
   StatusWriteTx( Status );

   // if new incoming status available, process it.
   if( StatusReadRx( &Status ) == STATUS_NEW_VALUE )
   {
    if( !( Status & STATUS_OFF_HOOK_N ) )
     LINE_INTF_OFF_HOOK_N = ASSERT_N;
    else LINE_INTF_OFF_HOOK_N = !ASSERT_N;
   }
  }
```

```
/*****************************************************************************/
/*                                                                           */
/* Millenium TelePath Line-side Device f/ware                                */
/* ------------------------------------                                      */
/*                                                                           */
/* This firmware implements the line-side functionality of the               */
/* Millenium TelePath wireless POTS device.                                  */
/*                                                                           */
/*****************************************************************************/
#include <..\std_inc\cc_std.h>     // standard CC types
#include <..\std_inc\compiler.h>   // compiler specific defines
#include <..\chnl\chnl.h>          // channel selection module
#include <..\mb15e03\mb15e03.h>    // IRF9xxx radio frequency synth module
#include <..\lineintf\lineintf.h>  // line interface module
#include <..\ds1706\ds1706.h>      // CPU-supervisor module
#include <..\aic9001\aic9001.h>    // DSSS transceiver module
#include <..\pkt\pkt.h>            // Config port packet module
#include <..\m93x6\m93x6.h>        // M93x6 serial EEPROM module

void main( void )
{
 DS1706_Reset();
 M93x6_Init();
 PktInit();
 ChnlInit();
 LineIntfInit();
 MB15E03_Init();
 AIC9001_Init();
 MB15E03_Mode( MASTER );
 AIC9001_Mode( MASTER );
 AIC9001_Connect();
 EnableInt();
 while( 1 )
 {
  LineIntfRun();
  DS1706_Reset();
  PktRun();
 }
}
```

```
#ifndef MB15E03_H
#define MB15E03_H

/*****************************************************************************/
/*                                                                           */
/* Fujitsu MB15E03 single VCO Phase-locked Loop Module                       */
/* ---------------------------------------------                             */
/*                                                                           */
/* File: ..\mb15e03\mb15e03.h                                                */
/* Date: 99.09.20                                                            */
/* By:   Arch                                                                */
/* Description:                                                              */
/*                                                                           */
/* Fujitsu MB15E03 single VCO Phase-locked Loop Module provides              */
/* an interface to the MB15E03 device used in the ALFA IRF9xxx               */
/* radio module.  A call to MB15E03_Init initializes the device              */
/* I/O.  A call to MB15E03_Mode configures the device registers              */
/* for the appropriate transmit and receive frequencies.                     */
/*                                                                           */
/* Reference Clock     = 16.384 MHz                                          */
/* Resolution          = 256.0 kHz                                           */
/* IF                  = 44.0 MHz                                            */
/* Prescaler           = 64                                                  */
/*                                                                           */
/*****************************************************************************/

#include <..\mcu_def\i8xc32.h>
#include <..\aic9001\aic9001.h>

#define MB15E03_SELECT   AIC9001_Mode( SLAVE )
#define MB15E03_DESELECT AIC9001_Mode( MASTER )

// TYPES
typedef enum              // CNT bit type for register selection.
{
 COUNTER = 0x0000,
 REF_COUNTER = 0x8000
} MB15E03_RegType;

typedef struct            // Control bits byte for reference divider register.
{
 unsigned :12;            //                    LSB
 unsigned SW:1;           // * Divide ratio for the prescaler (64/65 or 128/129).
 unsigned FC:1;           // * Phase control for the phase comparator.
 unsigned LDS:1;          // * LD/fout signal select.
```

```
 unsigned CS:1;            // * Charge pump current.    MSB
} MB15E03_CntrlBitsType;

typedef struct             // Programmable reference counter type.
{
 MB15E03_CntrlBitsType CntrlBits;// * control bits
 Tmedium R;                // * ratio
} MB15E03_RefCounterType;

typedef struct             // Programmable counter type.
{
 Tmedium N;                // * ratio
 Tsmall A;                 // * swallow counter
} MB15E03_CounterType;

void MB15E03_Init( void );
void MB15E03_Mode( M_SN_Type Data );

#endif
```

```
#ifndef MSG_H
#define MSG_H

// DEFINITIONS
#define Msg      (*(MsgType *)&PktBuf)
#define MsgLength PktLength
#define MsgRead   PktRead
#define MsgWrite  PktWrite

// TYPES
typedef enum { REQUEST, REPLY } MsgKindType;
typedef enum { IDATA, EE } MsgSpaceType;
typedef enum { READ, WRITE } MsgDirType;
typedef Tsmall MsgChkSumType;
typedef struct
{
 MsgChkSumType ChkSum;
 MsgKindType Kind:1;
 MsgSpaceType Space:1;
 MsgDirType Dir:1;
 Tmedium Adrs;
 Tsmall Data;
} MsgType;

MsgChkSumType MsgComputeChkSum( void );

#endif
```

```
// INCLUDES
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\i8051sci\i8051sci.h>
#include <..\m93x6\m93x6.h>
#include <..\pkt\pkt.h>
#include <..\mcu_def\i8xc32.h>
#include <..\aic9001\aic9001.h>
#include <..\msg\msg.h>

// FUNCTIONS
MsgChkSumType MsgComputeChkSum( void )
{
 Tsmall Count;
 Tsmall ChkSum = 0;

 for( Count = sizeof( MsgChkSumType ); Count < sizeof( MsgType ); Count++ )
  ChkSum ^= ((Tsmall *)&Msg)[Count];
 return( ChkSum );
}

void MsgRead( void )
{
 if( ( MsgLength != sizeof( MsgType ) ) ||
    ( Msg.ChkSum != MsgComputeChkSum() ) ||
    ( Msg.Kind != REQUEST ) )
  return;
 if( Msg.Space == IDATA )
 {
  if( Msg.Dir == READ )
   Msg.Data = *(Tsmall idata *)Msg.Adrs;
  else *(Tsmall idata *)Msg.Adrs = Msg.Data;
 }
 else
 {
  if( AIC9001_RST2_N == !ASSERT_N )
   IE_EX1 = CLEAR;
  if( Msg.Dir == READ )
   Msg.Data = M93x6_Read( Msg.Adrs );
  else M93x6_Write( Msg.Adrs, Msg.Data );
  if( AIC9001_RST2_N == !ASSERT_N )
   IE_EX1 = SET;
 }
Msg.Kind = REPLY;
Msg.ChkSum = MsgComputeChkSum();
```

```
  MsgLength = sizeof( MsgType );
  MsgWrite();
}
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\m93x6\m93x6.h>
#include <..\msg\msg.h>
#include <..\pkt\pkt.h>

void main( void )
{
 PktInit();
 M93x6_Init();
 EnableInt();

 while(1)
 {
  PktRun();
 }
}
```

```
/*****************************************************************************/
/*                                                                           */
/* Millenium TelePath Phone-side Device f/ware                               */
/* ------------------------------------                                      */
/*                                                                           */
/* This firmware implements the phone-side functionality of the              */
/* Millenium TelePath wireless POTS device.                                  */
/*                                                                           */
/*****************************************************************************/
#include <..\std_inc\cc_std.h>     // standard CC types
#include <..\std_inc\compiler.h>   // compiler specific defines
#include <..\chnl\chnl.h>          // channel selection module
#include <..\mb15e03\mb15e03.h>    // IRF9xxx radio frequency synth module
#include <..\hc55181\hc55181.h>    // SLIC module
#include <..\ds1706\ds1706.h>      // CPU-supervisor module
#include <..\aic9001\aic9001.h>    // DSSS transceiver module
#include <..\pkt\pkt.h>            // Config port packet module
#include <..\m93x6\m93x6.h>        // M93x6 serial EEPROM module

void main( void )
{
 DS1706_Reset();
 M93x6_Init();
 PktInit();
 ChnlInit();
 HC55181_Init();
 MB15E03_Init();
 AIC9001_Init();
 MB15E03_Mode( SLAVE );
 AIC9001_Mode( SLAVE );
 AIC9001_Connect();
 EnableInt();
 while( 1 )
 {
  HC55181_Run();
  DS1706_Reset();
  PktRun();
 }
}
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\queue\queue.h>

/*****************************************************************************/
/*                                                                           */
/* QueueInit()                                                               */
/* -----------                                                               */
/*                                                                           */
/* This function initializes a queue object.  It zeros both the             */
/* Head and Tail indices.                                                    */
/*                                                                           */
/*****************************************************************************/
void QueueInit( void *Q )
{
 ((QueueType *)Q)->Head = 0;
 ((QueueType *)Q)->Tail = 0;
}
```

```
#ifndef PKT_H
#define PKT_H

#include <..\std_inc\cc_std.h>

#define PKT_MAX_BYTES 5

extern Tsmall PktBuf[PKT_MAX_BYTES];
extern Tsmall PktLength;

void PktInit( void );
void PktRun( void );
void PktRead( void );
void PktWrite( void );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\i8051sci\i8051sci.h>
#include <..\pkt\pkt.h>

#define PKT_USE_VALUE_CHAR '\\'
#define PKT_END_OF_PKT_CHAR ' '
#define PKT_BUILDING 0x01
#define PKT_USE_VALUE 0x02

Tsmall PktLength;
Tsmall PktBuf[PKT_MAX_BYTES];

void PktInit( void )
{
 i8051SciInit();
}

void PktRun( void )
{
 static Tsmall Status = 0x00;
 Tsmall RxByte;

 if( !i8051SciRead( &RxByte ) )
  return;
 if( !(Status & PKT_BUILDING) )
 {
  Status |= PKT_BUILDING;
  PktLength = 0;
 }
 if( Status & PKT_USE_VALUE )
 {
  Status &= ~PKT_USE_VALUE;
  if( PktLength < PKT_MAX_BYTES )
   PktBuf[PktLength++] = RxByte;
  return;
 }
 switch( RxByte )
 {
  case PKT_USE_VALUE_CHAR:
   Status |= PKT_USE_VALUE;
   break;
  case PKT_END_OF_PKT_CHAR:
   Status &= ~PKT_BUILDING;
   PktRead();
```

```
      break;
    default:
      if( PktLength < PKT_MAX_BYTES )
        PktBuf[PktLength++] = RxByte;
  }
}

void PktWrite( void )
{
 Tsmall Count;

 for( Count = 0; Count < PktLength; Count++ )
 {
  switch( PktBuf[Count] )
  {
    case PKT_USE_VALUE_CHAR:
    case PKT_END_OF_PKT_CHAR:
      i8051SciWrite( PKT_USE_VALUE_CHAR );
    default:
      i8051SciWrite( PktBuf[Count] );
  }
 }
 i8051SciWrite( PKT_END_OF_PKT_CHAR );
}
```

```
#ifndef QUEUE_H
#define QUEUE_H

/*****************************************************************************/
/*                                                                           */
/* Queue Module                                                              */
/* ------------                                                              */
/*                                                                           */
/* File: ..\Queue\Queue.h                                                    */
/* Date: 99.09.20                                                            */
/* By:   Arch                                                                */
/* Description:                                                              */
/*                                                                           */
/* The Queue Module provides queue services for 8-bit data.                  */
/* Queue objects are created at compile-time by invoking the                 */
/* QUEUE_CREATE macro. This allocates the required memory and                */
/* declares the read and write functions within the user's code.             */
/* This unusual approach was selected to avoid passing queue                 */
/* pointers to generic read/write routines resulting in slow code.           */
/*                                                                           */
/* Invocation of QUEUE_CREATE( <name>, <size> ) creates a                    */
/* queue object called <name> that can hold a maximum of <size>              */
/* characters.  Memory is allocated and functions <name>Write                */
/* and <name>Read are declared.                                              */
/*                                                                           */
/* A queue object must be initialized before use by calling                  */
/* QueueInit( &<name> ).                                                     */
/*                                                                           */
/* Data are written to the queue by calling <name>Write( <data> ).           */
/* Data are read from the queue by calling <name>Read( &<data> ).            */
/* Both functions return booleans to indicate the success of the             */
/* respective operation.                                                     */
/*                                                                           */
/*****************************************************************************/

#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>

typedef struct
{
 Tsmall Head;
 Tsmall Tail;
 Tsmall Data[1];
} QueueType;
```

```
#define QUEUE_CREATE( Q, Size )                          \
Tsmall Q[( sizeof( QueueType ) + ( sizeof( Tsmall ) * 10 ) )]; \
Tbool Q##Write( Tsmall Data )                            \
{                                                        \
 Tsmall TailCopy = ((QueueType *)Q)->Tail;               \
                                                         \
 TailCopy++;                                             \
 if( TailCopy > Size )                                   \
  TailCopy = 0;                                          \
 if( TailCopy == ((QueueType *)Q)->Head )                \
  return( FALSE );                                       \
 ((QueueType *)Q)->Data[((QueueType *)Q)->Tail] = Data;  \
 ((QueueType *)Q)->Tail = TailCopy;                      \
 return( TRUE );                                         \
}                                                        \
Tbool Q##Read( Tsmall *Data )                            \
{                                                        \
 if( ((QueueType *)Q)->Head == ((QueueType *)Q)->Tail )  \
  return( FALSE );                                       \
 *Data = ((QueueType *)Q)->Data[((QueueType *)Q)->Head]; \
 ((QueueType *)Q)->Head++;                               \
 if( ((QueueType *)Q)->Head > Size )                     \
  ((QueueType *)Q)->Head = 0;                            \
 return( TRUE );                                         \
}

void QueueInit( void *Q );

#endif
```

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\aic9001\aic9001.h>
#include <..\status\status.h>

/*****************************************************************************/
/*                                                                           */
/* StatusWriteTx()                                                           */
/* ---------------                                                           */
/*                                                                           */
/* This function writes status data to the AIC9001 for transmission. */
/*                                                                           */
/*****************************************************************************/
void StatusWriteTx( Tsmall TxStatus )
{
 AIC9001_TxStatus = TxStatus;
}

/*****************************************************************************/
/*                                                                           */
/* StatusReadRx()                                                            */
/* --------------                                                            */
/*                                                                           */
/* This function reads, filters, and returns receive status data.    */
/*                                                                           */
/* It returns one of two states to describe the nibble:              */
/* * STATUS_UNCHANGED   indicates that the nibble presented has      */
/*                 been presented before as new data,                */
/* * STATUS_NEW_VALUE   indicates that the nibble has not before     */
/*                 been presented.                                   */
/*                                                                           */
/*****************************************************************************/
StatusStateType StatusReadRx( Tsmall *RxStatus )
{
 static StatusStateType State = STATUS_UNCHANGED;
 static Tsmall Old = 0xFF;
 Tsmall New;

// Try
 if( !AIC9001_RxStatusRead( &New ) )// if no RxStatus to receive...
  goto Exception0;              //  do exception.
 if( New == Old )               // if unchanged...
  goto Exception1;              //  do exception.
 if( ++State != STATUS_NEW_VALUE ) // if bouncing...
  goto Exception0;              //  do exception.
```

```
  State = STATUS_UNCHANGED;        // else
  Old = New;                       //  update Old
  *RxStatus = New;                 //  return new
  return( STATUS_NEW_VALUE );      //  return new value status.

// Catch
Exception1:
  State = STATUS_UNCHANGED;
Exception0:
  *RxStatus = Old;
  return( STATUS_UNCHANGED );
}
```

```
#ifndef STATUS_H
#define STATUS_H

/*****************************************************************************/
/*                                                                           */
/* Status Module                                                             */
/* -------------                                                             */
/*                                                                           */
/* File: ..\status\status.h                                                  */
/* Date: 99.09.20                                                            */
/* By:   Arch                                                                */
/* Description:                                                              */
/*                                                                           */
/* The Status Module provides an interface to read and write                 */
/* status from and to the remote device.  Status data comprise               */
/* the four bits available in the AIC9001 spread-spectrum                    */
/* transceiver.  Status written with the StatusWrite function                */
/* is store directly to the AIC9001 module for transmission.                 */
/* It is not buffered in any way.  This value will be repeatedly             */
/* transmitted, once per frame, until it is rewritten with                   */
/* a new value by a subsequent call.                                         */
/*                                                                           */
/* Received status is read by a call to StatusRead.  The stream              */
/* of received commands are queued and filtered to "debounce"                */
/* any noise that may have been introduced in the transmission.              */
/* When a change in the received status passes through the filter,           */
/* the function returns a STATUS_NEW result, else it returns                 */
/* a STATUS_UNCHANGED result.                                                */
/*                                                                           */
/*****************************************************************************/

#define STATUS_COUNT 2

typedef enum
{
 STATUS_UNCHANGED,
 STATUS_WAITING,
 STATUS_NEW_VALUE = STATUS_WAITING + STATUS_COUNT - 1
} StatusStateType;

#define STATUS_OFF_HOOK_N   0x80
#define STATUS_TR_REVERSED_N 0x80
#define STATUS_RING_N       0x40

void StatusWriteTx( Tsmall TxStatus );
```

StatusStateType StatusReadRx(Tsmall *RxStatus);

#endif

```
#include <..\std_inc\cc_std.h>
#include <..\std_inc\compiler.h>
#include <..\mcu_def\i8xc32.h>
#include <..\i8051sci\i8051sci.h>

void PutString( char *String, Tsmall Len )
{
 while( Len-- && i8051SciWrite( *String++ ) );
}

int main( void )
{
 Tsmall Data;

 i8051SciInit();
 EnableInt();
 PutString( " abcdef-", 8 );
 while( 1 )
 {
  if( i8051SciRead( &Data ) )
  {
   i8051SciWrite( Data );
   if( Data == '#')
    break;
  }
 }
 PutString( "\n\rBye!", 8 );
}
```